

HP iLO Chassis Management IPMI User Guide

Abstract

This document provides customers with information on the implementation of the Intelligent Platform Management Interface in HP Moonshot iLO Chassis Management Firmware, including the available commands.



Notices

Confidential computer software. Valid license from HP required for possession, use, or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

Contents

1	Introduction and key concepts.....	7
	Overview.....	7
	Sensor Data Model.....	7
	Sensor owner identification.....	7
	Sensor type code.....	8
	System event log and event messages.....	8
	SDR repository.....	10
	SDR formats.....	10
	Reading the SDR repository and device SDR repositories.....	11
	FRU.....	11
	FRU inventory device.....	12
	Standardized timers.....	12
	Watchdog timer.....	12
	POH counter.....	12
	Timestamp format.....	12
2	The virtual topology of the Moonshot 1500 CM module.....	14
3	Discovering managed entities using IPMITool.....	17
4	IPMITool.....	18
	Moonshot IPMITool support — out of band.....	18
	Interfaces.....	19
	System Interface.....	19
	LANPlus Interface.....	19
	Features.....	20
	Events.....	21
	Inventory.....	22
	Chassis management.....	22
	Synopsis.....	22
	IPMITool Raw command syntax and example.....	24
5	Command specification.....	25
	Standard command specification.....	29
	Global commands.....	29
	Get device ID command.....	29
	Cold reset command	32
	Warm reset command.....	33
	Get self test results command	33
	Get ACPI power state command	34
	Broadcast get device ID command	35
	IPMI messaging support commands.....	36
	Set BMC global enables command	36
	Get BMC global enables command	37
	Clear message flags command	37
	Get message flags command	38
	Enable message channel receive command	38
	Get message command	39
	Send message command.....	42
	Get system GUID command	44
	Set system info parameters command	45
	Get system info parameters command.....	46
	Master write-read command.....	49

Get channel authentication capabilities command.....	50
Get Channel Cipher Suites Command.....	52
Cipher suite records.....	54
Cipher suite ID numbers.....	55
Set session privilege level command	56
Close session command.....	57
Get session info command	57
Get AuthCode command	59
Set channel access command	60
Get channel access command	62
Get channel info command	63
Set user access command.....	65
Get user access command.....	66
Set user name command	68
Get user name command.....	68
Set user password command.....	69
RMCP+ support and payload commands.....	70
Activate payload command.....	71
Deactivate payload command.....	73
Suspend/resume payload encryption command.....	74
Set channel security keys command.....	75
Get system interface capabilities command.....	77
Get payload activation status command.....	78
Get payload instance info command.....	79
Set user payload access command.....	79
Get user payload access command.....	80
Get channel payload support command.....	81
Get channel payload version command.....	82
IPMI LAN Device Commands.....	83
Set LAN configuration parameters command.....	83
Get LAN configuration parameters command.....	83
SOL commands.....	92
Set SOL configuration parameters command.....	92
Get SOL configuration parameters command.....	93
MC watchdog timer commands.....	96
Watchdog timer actions.....	96
Watchdog timer use field and expiration flags.....	97
Using the timer use field and expiration flags.....	97
Watchdog timer event logging.....	97
Pre-timeout interrupt.....	98
Pre-timeout interrupt support detection.....	98
BIOS support for watchdog timer.....	98
Reset watchdog timer command	98
Set watchdog timer command	98
Get watchdog timer command	100
Chassis commands.....	102
Get chassis capabilities command	102
Get chassis status command	103
Chassis control command	104
Chassis identify command	105
Set power restore policy command	106
Set system boot options command	107
Get system boot options command.....	107
Get POH counter command	111
Event commands.....	112

Set event receiver command.....	112
Get event receiver command.....	113
Platform event message command.....	113
SEL commands.....	114
SEL device commands.....	114
Get SEL info command.....	114
Reserve SEL command.....	115
Get SEL entry command.....	116
Add SEL entry command.....	116
Clear SEL.....	117
SEL record type ranges.....	117
Get SEL time command.....	118
Set SEL time command.....	118
SDR repository device commands.....	118
SDR record IDs.....	118
Get SDR repository info command.....	119
Get SDR repository allocation info command.....	120
Reserve SDR repository command.....	120
Reservation restricted commands.....	121
Reservation cancellation.....	121
Get SDR command.....	121
Add SDR command.....	122
Delete SDR command.....	123
Clear SDR repository command.....	123
Run initialization agent command.....	124
FRU inventory device commands.....	124
Get FRU inventory area info command.....	125
Read FRU data command.....	125
Write FRU data command.....	126
Sensor Device Commands.....	126
Get device SDR info command.....	126
Get device SDR command.....	127
Reserve device SDR repository command.....	128
Get sensor thresholds command.....	128
Get sensor reading command.....	129
DCMI specific commands.....	130
Get DCMI capability info command.....	131
Get asset tag command.....	133
Get DCMI sensor info command.....	134
Set asset tag command.....	135
Management controller ID string.....	135
Get controller ID string command.....	136
Set controller ID string command.....	136
PICMG specific commands.....	137
Get PICMG properties command.....	137
Get address info command.....	137
FRU inventory device lock control command.....	138
6 IPMI Messaging and Interfaces.....	140
System Interfaces.....	140
Message interface description.....	141
IPMI Messaging Interfaces.....	141
Network function codes.....	141
Completion codes.....	142
Channel Model, Authentication, Sessions, and Users.....	144

Channel numbers.....	145
Logical channels.....	145
Channel Privilege Levels.....	145
Users & Password support.....	146
IPMI sessions.....	146
Session-less connections.....	147
Session inactivity timeouts.....	147
System interface messaging.....	147
Bridging.....	148
MC LUN 10b.....	148
Send Message command with response tracking.....	149
Bridged Request Example.....	149
IPMB access via master write-read command.....	151
MC IPMB LUNs.....	151
Sending Messages to IPMB from system software.....	152
Keyboard Controller Style Interface.....	153
KCS Interface/MC LUNs.....	153
KCS Interface-MC Request message format.....	153
MC-KCS Interface Response Message format.....	153
LAN Interface.....	154
Remote Management Control Protocol (RMCP).....	154
RMCP port numbers.....	154
RMCP Message Format.....	155
Serial Over LAN (SOL).....	156
7 Support and other resources.....	157
Information to collect before contacting HP.....	157
How to contact HP.....	157
HP authorized resellers.....	157
Related information.....	157
A Command Assignments.....	159
B Verbose output examples.....	164
Glossary.....	186
Index.....	188

1 Introduction and key concepts

Overview

The term Intelligent Platform Management (IPMI), refers to autonomous monitoring and recovery features implemented directly in platform management hardware and firmware. The key characteristic of Intelligent Platform Management is that inventory, monitoring, logging, and recovery control functions are available independently of the main processors, BIOS, and operating system. Platform management functions are available even when the system is in a powered down state.

IPMI capabilities are a key component in providing enterprise-class management for HA systems. Platform status information is obtained and recovery actions initiated under situations where system management software and normal “in-band” management mechanisms are unavailable.

The independent monitoring, logging, and access functions available through IPMI provide a level of manageability built-in to the platform hardware. This supports systems with no system management software available for the particular operating system, or the end-user who elects not to load or enable the system management software.

NOTE: The HP Moonshot-45G Switch Module does not support IPMI. Only the HP Moonshot iLO Chassis Management Firmware supports IPMI.

Sensor Data Model

The IPMI Sensor Model provides access to monitored information including temperatures, voltages, and fan status. Instead of providing direct access to the monitoring hardware, IPMI provides access by abstracted sensor commands, such as the `Get Sensor Reading` command, implemented via a management controller. This approach isolates software from changes in the platform management hardware implementation. Sensors return analog or discrete readings and events are either discrete or threshold-based. Sensors are classified according to:

- Type of readings
- Type of events

Event types, sensor types, and monitored entities are represented using numeric codes defined in the IPMI specification. IPMI avoids reliance on strings for management information and using numeric codes facilitates internationalization, automated handling by higher level software, and reduces management controller code and data space requirements.

Sensor owner identification

The definition for the Request/Response identifier, Requester’s ID, and Responder’s ID are specific to the particular messaging interface used. However, the SDR and SEL must contain information to identify the `owner` of the sensor. For management controllers, a slave address and LUN identify the owner of a sensor on the IPMB. For system software, a software ID identifies the sensor owner. These fields are used in event messages, where events from management controllers or the IPMB are identified by an eight-bit field where the upper 7 bits represent the slave address or the system software ID. The least significant bit is 0 if the value represents a slave address and 1 if the value represents a system software ID.

Sensor number is not part of the sensor owner ID, but is a separate field used to identify a particular sensor associated with the sensor owner. This combination of sensor owner ID and sensor number uniquely identify a sensor in the system.

Table 1 Sensor owner ID and sensor number field definition

IPMB Sensor Owner ID	System Sensor Owner ID
7:1 slave address (7 bits)	system software ID (7 bits)

Table 1 Sensor owner ID and sensor number field definition *(continued)*

IPMB Sensor Owner ID	System Sensor Owner ID
0 0b (ID is a slave address)	0 1b (ID is a software ID)
LUN (2 bits)	sensor number (8 bits, FFh = reserved)
sensor number (1 bit, FFh = reserved)	
In Moonshot: 0x82 IPMB address	This only appears in the system node SEL, where the it is logged by the host system

Sensor type code

Each sensor has a sensor type code and are defined in “[Some Moonshot sensor type codes](#)” (page 8). Sensor type codes are used both in SDRs and event messages. An example of a sensor type code is code 0x1, which indicates a temperature sensor.

Table 2 Some Moonshot sensor type codes

Sensor type	Sensor type code	Reading type code
Temperature	0x1	0x1
Fan	0x4	0xA
Fan redundancy	0x4	0xB
PICMG IPMB0 Physical Link	0xF1	0x6F
Health LED	0xC0	0x71
UID LED	0xC0	0x70
Power supply	0x8	0x6F
Power supply redundancy	0x8	0xB

For a complete listing of sensor type codes, see the IPMI specification available at:

<http://www.intel.com/content/www/us/en/servers/ipmi/second-gen-interface-spec-v2.html>

System event log and event messages

The MC provides a centralized, non-volatile SEL. Having the SEL and logging functions managed by the MC helps ensure that post-mortem logging information is available should a failure occur that disables the systems processor(s).

A set of IPMI commands allows the SEL to be read and cleared, and for events to be added to the SEL. The common request message used for adding events to the SEL is an event message. Event messages are sent to the MC via the IPMB providing the mechanism for satellite controllers to detect events and log them into the SEL. The controller that generates an event message to another controller via IPMB is the IPMB Event Generator. The controller receiving event messages is the IPMB Event Receiver.

In Moonshot, event messages are sent to the zone manager by each cartridge and chassis controller. There are two event logs, the SEL and the IML. There are OEM specific commands for obtaining and displaying these logs. For more information, see the related commands in “[Command specification](#)” (page 25).

Event messages are special messages sent by management controllers when they detect significant or critical system management events. This includes messages for events such as:

- temperature threshold exceeded
- voltage threshold exceeded
- power fault

The event message generator notifies the system by sending an Event Request Message to the event receiver device.

When the event receiver gets a valid event message, it sends a response message to the event message generator which is typically transferred to the SEL. The event receiver does not interpret event messages so that new event message types can be added into the system without impacting event receiver implementation.

SEL commands — The SEL is a non-volatile repository for system events and some system configuration information. The SEL device access the commands sent by the SEL. Event messages when received by the event receiver device are written to the SEL.

Table 3 SEL event records

Byte	Field	Description
1 2	Record ID	ID used for SEL record access. The Record ID values 0000h and FFFFh have special meaning in the Even Access commands and must not be used as Record ID values for stored SEL event records.
3	Record type	[7:0] — Record type 02h = system event record C0h-DFh = OEM timestamped, bytes 8–16 OEM defined E0h-FFh = OEM non-timestamped, bytes 4–16 OEM defined
4 5 6 7	Timestamp	Time when event was logged. LS byte first.
8 9	Generator ID	RqSA & LUN if event was generated from IPMB. Software ID if event was generated from system software. <u>Byte 1</u> [7:1] — 7-bit I ² C. Slave address, or 7-bit system software ID [0] 0b = ID is IPMB slave address 1b = System software ID <u>Byte 2</u> [7:4] — Channel number. Channel that received the event message 0h if the event message was received via the system interface, primary IPMB, or internally generated by the MC. [3:2] — Reserved. Write as 00b. [1:0] — IPMB device LUN if byte 1 holds slave address, otherwise 00b.
10	EvM Rev	Event message format version (=04h for events in this specification, 03h for IPMI v1.0 event messages). ¹
11	Sensor type	Sensor type code for sensor that generated the event.
12	Sensor #	Number of sensor that generated the event.
13	Event dir 1 Event type	<u>Event dir</u> [7] — 0b = Assertion event. 1b = Deassertion event.

Table 3 SEL event records *(continued)*

Byte	Field	Description
		<u>Event type</u> Type of trigger for the event, such as, a critical threshold going high or state asserted. Also indicates class of the event. Example: discrete, threshold, or OEM. The event type field is encoded using the event/reading type code.
14	Event data 1	Event request message, event data field contents.
15	Event data 2	Event request message, event data field contents.
16	Event data 3	Event request message, event data field contents.

¹ The MC must accept Platform Event request messages that are in IPMI v1.0 format (EvM Rev=03h) and log them as IPMI v1.5/v2.0 records by setting the EVMRev field to 04h and setting the channel number in the generator ID field appropriately for the channel that received the event.

SDR repository

With IPMI's extensibility and scalability each platform implementation can have a different population of management controllers and sensors, and different event generation capabilities. IPMI allows system management software to retrieve information from the platform and automatically configure itself to the platform's capabilities, enabling the use of plug and play, platform-independent instrumentation software.

Information that describes the platform management capabilities is provided via two mechanisms:

- Capability commands — these are commands within the IPMI command set that return information on other commands and functions that the controller can handle.
- SDRs — these contain information about the type and number of sensors in the platform, sensor threshold support, event generation capabilities, and sensor type readings.

The primary purpose of SDRs is to describe the sensor configuration of the platform management subsystem to system software. SDRs also include records describing the number and type of devices connected to the system's IPMB, records that describe the location and type of FRU Devices (devices that contain field replaceable unit information).

SDRs are kept in a single, centralized, non-volatile storage area managed by the MC. This storage area is the SDR repository. In Moonshot, the SDRR is kept by the zone manager; the remaining controllers have device SDRs. Additional device SDRs are kept at the cartridge and system node level. All SDR repositories provide a mechanism for information to be obtained independently from the controller by the BIOS system management or remote management.

SDR formats

The general SDR format consists of three major components: the record header, record key fields, and the record body. To save space, sensors that only generate events do not require SDRs, in addition, generic system management software does not access sensors unless they are reported by SDRs.

Table 4 Sensor data record formats

Record header	Record Key fields	Record body
Record ID — a value used for accessing sensor data records.	The record key bytes are the contiguous bytes following the record header. The number of bytes vary according to record type. Together, they make up a set of unique fields for a given record specifying location (for example, slave address, LUN and Bus ID) and sensor number.	Contains specific information to the sensor data record
SDR version — version number of the SDR specification.		
Record type — a number representing the type of record. Example, 01h = 8-bit sensor with thresholds.		
Record length — the number of bytes of data following the record length field.		

Reading the SDR repository and device SDR repositories

An application that retrieves records from the SDR repository must first read them sequentially using the `Get SDR` command. This command returns the requested record and the record ID of the next SDR in sequence.

NOTE: Record IDs are not required to be sequential or consecutive and applications should not assume that SDR record IDs follow any particular numeric ordering.

Retrieve succeeding records by issuing the `Get SDR` or `Get device SDR` command using the next record ID returned in the previous response. This is continued until the `End of Record ID` is encountered.

Once all the desired records have been read, the application can randomly access the records according to their Record ID. An application that seeks to access records randomly must save a data structure that retains the record key information according to the record ID.

❗ **IMPORTANT:** Record IDs may change with time, it is important for applications to first verify that the Record Key information matches the record retrieved.

If the record ID is no longer valid for a record key, then, access the SDR records again as described above until the record matches the record key.

An application can tell whether records have changed by examining the most recent addition timestamp using the `Get SDR repository info` or `Get device SDR repository info` command, depending on the zone in which the command is issued.

If the record information has changed, an application does not need to list out the entire contents of all records. The `Get SDR` or `Get device SDR` allows a partial read of the SDR. Thus, an application can search for a given `Record Key` by just retrieving that portion of the record.

FRU

The IPMI specifications include support for storing and accessing multiple sets of non-volatile FRU data for different modules in the system. An enterprise-class system typically has FRU information for each major system board such as the processor board, memory board or I/O board. FRU data includes serial number, part number, model, and asset tag.

IPMI FRU information is accessible via the IPMB and management controllers. The information can be retrieved at any time, independent of the main processor, BIOS, system software, or OS, via out-of-band interfaces, such as the ICMB, a remote management card, or other device connected to the IPMB. FRU information is still available when the system is powered down.

With these capabilities FRU information is available even under failure conditions when access mechanisms that rely on the main processor are unavailable. This facilitates the creation of automated remote inventory and service applications. IPMI does not seek to replace other FRU or

inventory data mechanisms such as those provided by SM BIOS, and PCI vital product data. Rather, IPMI FRU information is typically used to complement that information or provide information access out-of-band or under *system down* conditions.

IPMI provides FRU information in two ways: via a management controller, or via FRU SEEPROMs. FRU information that is managed by a management controller is accessed using IPMI commands. This isolates software from direct access to the non-volatile storage device, allowing the hardware implementor to utilize whatever type of non-volatile storage required.

FRU inventory device

The FRU inventory device contains information such as the serial number, part number, asset tag, and short descriptive string for the FRU. The contents of a FRU inventory record are specified in the platform management FRU information storage definition.

The FRU inventory device is a logical device and is not necessarily implemented as a separate physical device. This device contains the SDR repository device and typically holds FRU inventory information for the main system board and chassis. In addition, there may be a separate FRU inventory device that provides access to the FRU information for a replaceable module such as a memory module.

FRU devices can be located either behind a management controller or directly on the IPMB. The sensor data records include a FRU device locator record that tells software where the device is located and the type of commands required to access the FRU device. FRU devices can be located in three types of location:

- Behind a management controller and accessed using Read/Write FRU data commands. Multiple FRU devices can be behind a management controller.
- SEEPROM on a private bus behind a management controller. These devices are accessed using Master Write-Read commands.
- SEEPROM on the IPMB. These devices are typically accessed using a Master Write-Read command to the IPMB via the MC.

Standardized timers

Watchdog timer

IPMI provides a standardized interface for a system watchdog timer that can also be used for BIOS, OS, and OEM applications. The timer can be configured to automatically generate selected actions when it expires; including power off, power cycle, reset, and interrupt. The timer function automatically logs the expiration event. Setting 0 for the timeout interval result causes the timeout action to be initiated immediately. This provides a means for devices on the IPMB, such as remote management cards, to use the watchdog timer to initiate emergency reset and other recovery actions dependent on the capability of the timer.

In Moonshot, watchdog timers are implemented by the system node controllers.

POH counter

The standardized power-on hours (POH) counter is optional. It returns a counter value proportional to the system operating power-on hours.

In Moonshot, the POH counter is implemented at the system node controller.

Timestamp format

A timestamp is a key component of event logging and tracking changes to the SDRs and the SDR repository.

Time is an unsigned, 32-bit value representing the local time as the number of seconds from 00:00:00, January 1, 1970.

The timestamps used for SDR and SEL records are specified in relative local time (that is, the difference between the timestamp does not include the GMT offset). Converting the timestamp to a GMT-based time requires adding the GMT offset for the system and is obtained from system software level interfaces. IPMI commands do not store or return GMT offset for the system. Applications may use ANSI C time standard library routines for converting the SEL timestamp into other time formats.

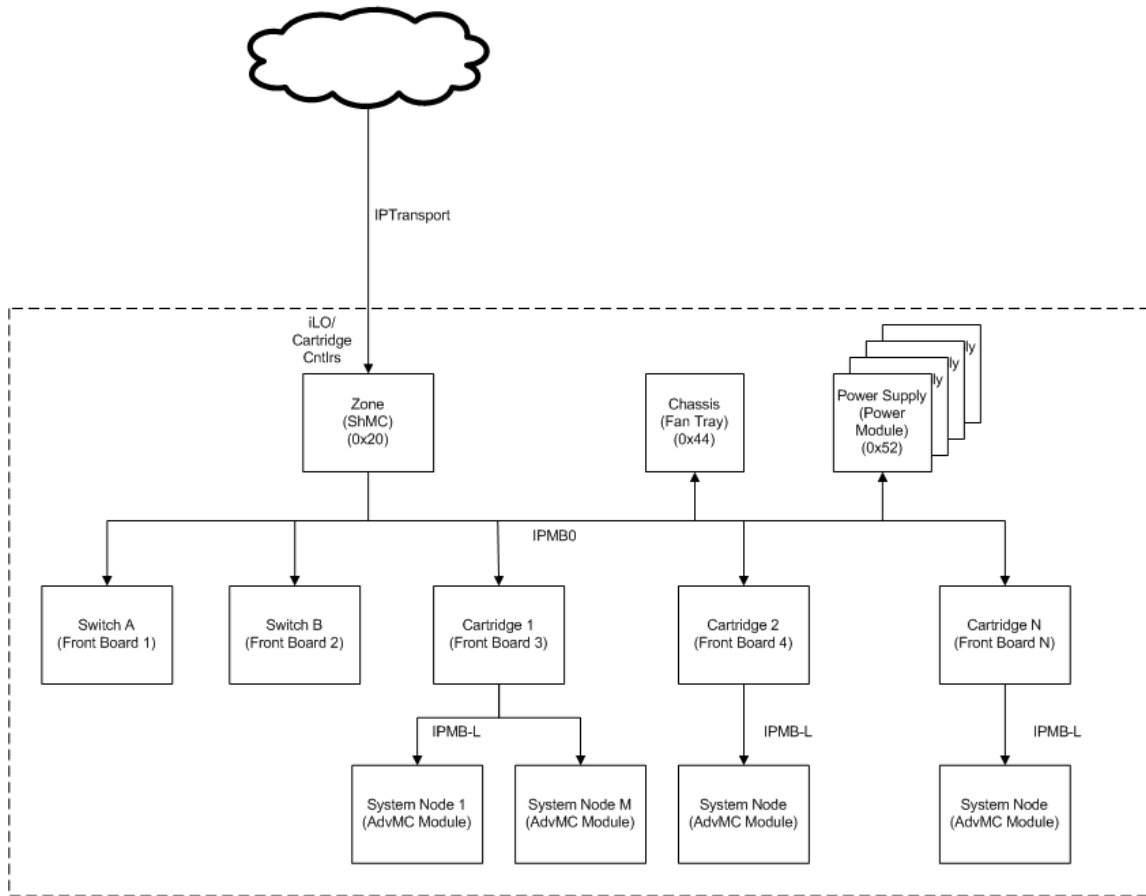
Special timestamp values

0xFFFFFFFF indicates an invalid or unspecified time value.

0x00000000 through 0x20000000 indicate events that occur after initialization of the SEL device up to when the timestamp is set with the system time value. These timestamp values are relative to the completion of the SEL devices initialization, and not January 1, 1970.

2 The virtual topology of the Moonshot 1500 CM module

Figure 1 Moonshot virtual IPMI topology



There are five virtual management controller (MC) entity types represented within the HP Moonshot 1500 Chassis Management module.

Table 5 (page 14) summarizes the various functions available in Moonshot and the virtual management controllers to which they apply.

Table 5 Moonshot virtual management controller functions

Function	Description	Applicable Virtual Management Controller				
		Zone	Chassis	Power Supply	Cart	Node
IPM Device	The MC must implement the mandatory IPM Device commands. If an IPMB is provided, the mandatory commands must be accessible from the IPMB unless otherwise noted.	x	x	x	x	x
System Interface	The implementation must provide MC access via one of the specified IPMI system interfaces.					x
SDR Repository	The MC must provide a SDR Repository to hold Sensor, Device Locator, and Entity Association records for all sensors in the platform management subsystem. This does not need to include SDRs for sensors that only generate events. If the SDR Repository is writable, it is recommended that at least 20% additional space is provided for add-in platform management extensions.	x				

Table 5 Moonshot virtual management controller functions *(continued)*

Function	Description	Applicable Virtual Management Controller				
		Zone	Chassis	Power Supply	Cart	Node
	The SDR Repository must be accessible via the system interface. If an IPMB is provided, the SDR Repository must be readable via that interface as well. SDR update via the IPMB interface is optional. SDR Repository access when the system is powered up or in ACPI 'S1' sleep is mandatory, but access when the system is powered -down or in a >S1 sleep state is optional.					
IPMB Interface	The IPMB is highly recommended, but optional. The MC must provide the system interface to the IPMB. If an IPMB is implemented, at least one of the specified IPMB connectors must be provided. Refer to the IPMB Protocol specification for connector definition. In addition the MC must implement a message channel that allows messages to be sent from the IPMB to the system interface, and vice-versa, and any other mandatory IPMB support functions and commands.	x	x	x	x	x
Watchdog Timer	The MC must provide the standardized Watchdog Timer interface, with support for system reset action. Certain functions within the Watchdog Timer are optional. Refer to the sections on the Watchdog Timer for information.					x
Event Receiver	The MC must implement an Event Receiver function and accept Event Messages via the system interface. If an IPMB is provided, the Event Receiver function must also accept Event Messages from the IPMB. Event Receiver operation while the system is powered up or in ACPI 'S1' sleep is mandatory, but operation when the system is powered down or in a >S1 sleep state is optional.	x				x
SEL Interface	The MC must provide a System Event Log interface. The event log must hold at least 16 entries. SEL access must be provided via the system interface. The SEL must be fully accessible via all mandatory SEL commands through all supported interfaces to the MC whenever the system is powered up or in ACPI 'S1' sleep state. SEL read access is always mandatory whenever the MC is accessible, and through any interface that is operational, regardless of system power state.	x				x
FRU Inventory	The MC must provide a logical Primary FRU inventory device, accessible via the Write- and Read FRU Data commands. The FRU Inventory Device Info command must also be supported. It is highly recommended that all other management controllers also provide a Primary FRU inventory device. (This was optional in IPMI v1.0.)	x	x	x	x	x
Initialization Agent	The initialization agent function is one where the MC initializes event generation and sensors both internally and on other management controllers according to initialization settings stored in the SDR for the sensor.	x	x	x	x	x

Table 5 Moonshot virtual management controller functions *(continued)*

Function	Description	Applicable Virtual Management Controller				
		Zone	Chassis	Power Supply	Cart	Node
Sensors	The MC can provide sensors. A typical server MC would provide sensors for baseboard temperature, voltage, and chassis intrusion monitoring.	x	x	x	x	x
Internal Event Generation	The MC must generate internal events for the Watchdog Timer. It is highly recommended that sensors generate events to eliminate the need for system management software to poll sensors, and to provide "post-mortem" failure information in the SEL. Internal event generation for sensors is optional, but highly recommended - particularly for 'environmental' (e.g. temperature and voltage) sensors.	x	x	x	x	x
External Event Generation	The MC could be designed to accept the <i>Set Event Receiver</i> command to allow it to be set as an IPMB Event Generator and send its event messages to another management controller. This would primarily be used for development and test purposes.		x	x	x	
LAN Messaging	Ability for the MC to send and receive IPMI Messaging over LAN	x				
LAN Alerting	Ability to send an Alert over the LAN	Not implemented yet.				
Bridging Support	The ability to transfer IPMI request and response messages between two interfaces connected to the MC. The following support is required if the corresponding interfaces are supported: <ul style="list-style-type: none"> • LAN <--> IPMB • LAN <--> System Interface 	x	x	x	x	x
Platform Event Filtering (PEF) and Alert Policies	Ability for MC to perform a selectable action on a n event. This capability is mandatory if paging or alerting is supported. Certain actions within PEF are optional. Refer to the sections on PEF for information. The Alert action and Alert Policies are mandatory if serial/modem or LAN alerting is supported.	Not implemented yet.				

3 Discovering managed entities using IPMITool

Enter the `IPMITool sdr list all` command to show all management controller records, whether you are querying an SDRR from the Virtual Zone MC or you are querying a device SDR for Virtual Cartridge MC records. For example:

```
# ipmitool -l lanplus -H iloatx-gem-2c -U admin -P admin123 sdr list all
```

ZoMC	Static MC @ 20h	ok
254	Log FRU @FEh f0.60	ok
IPMB0 Phys Link	0x00	ok
ChasMgmtCtrlr1	Static MC @ 44h	ok
PsMgmtCtrlr1	Dynamic MC @ 52h	ok
PsMgmtCtrlr2	Dynamic MC @ 54h	ok
PsMgmtCtrlr3	Dynamic MC @ 56h	ok
PsMgmtCtrlr4	Dynamic MC @ 58h	ok
CaMC	Dynamic MC @ 82h	ok
CaMC	Dynamic MC @ A0h	ok
CaMC	Dynamic MC @ A6h	ok
CaMC	Dynamic MC @ DAh	ok
CaMC	Dynamic MC @ A4h	ok

Querying an SDRR from the Zone Management Controller

- Enter the `sdr list all` command to show all management controller records.
- Management controller records
 - Chassis controller – statically assigned, always should be present. Even though always assigned address 0x44, record should be parsed to learn address and channel number (IPMB=0).
 - Power supply controllers – full complement of records always present. Dynamic indication in record indicates to application that controller may or may not be present. When not present, the controller will “nak”. Even though always assigned address 0x52-0x58, records should be parsed to learn address and channel number (IPMB=0).
 - Cartridge controllers are dynamically added/deleted to the SDRR upon cartridge insertion/deletion. Possibility of “nak” condition during deletion transitions. Records should be parsed to learn addresses and channel number (IPMB=0).

Querying a device SDR for a Cartridge Management Controller

- Management controller records
 - System node controllers – statically assigned, always should be present. Number of system node controllers is dependent on cartridge type. Some cartridges may not contain host systems such as switches. Other cartridges may contain 1 or 4 system nodes. Records should be parsed for addressing and channel number routing. System nodes routed on channel 7 (IPMB-L).

You can learn the physical locations of managed entities by entering the `PICMG get address info` command. Use the IPMB address of the cartridge to get the physical address (slot number). You can also learn the chassis topology through chassis FRU serial number for Zones with the same chassis.

4 IPMItool

IPMI tool is a simple command-line interface to systems that support the IPMI v1.5 specification. It provides the ability to read the sensor data repository and print sensor values, display the contents of the system event log, print field replaceable unit information, read and set LAN configuration parameters, and perform remote chassis power control. It was originally written to take advantage of IPMI-over-LAN interfaces but is also capable of using the system interface as provided by a kernel device driver such as Open IPMI. IPMItool is available under a BSD-compatible license.

System Management Software is generally complex and makes platform management only part of a much larger management picture. However, many system administrators and developers rely on command-line tools that can be scripted and systems that can be micro-managed. IPMItool takes a different approach to SMS and provides a completely command-line oriented tool. Therefore, it is not designed to replace the Open IPMI library. Where possible, it supports printing comma-separated values for output to facilitate parsing by other scripts or programs. It is designed to run quick command response functions that can be as simple as turning the system on or off or as complex as reading in the sensor data records and extracting and printing detailed sensor information for each record.

Moonshot IPMItool support — out of band

Single-bridging out of band command

You must single-bridge out of band commands to reach specific chassis management controller or cartridge management controller to discover management controller addresses.

Enter the `sdr list all` command at the zone management controller to discover chassis and cartridge management controller addresses. For example,

- Chassis Controller
`-b 0 -t 0x44`
- Cartridge Controller Slot 1
`-b 0 -t 0x82`
- Cartridge Controller Slot 2
`-b 0 -t 0x84`

where:

- `-b <ipmi channel number>`
- `-t <target slave address>`

Double-bridging out of band commands

You must double-bridge out of band commands to reach specific system node management controllers to discover system node management controller addresses.

Enter the `sdr list all` at the cartridge management controller to discover system node management controller addresses. For example,

- –System Node Controller 2 (0x74) on Cartridge Controller Slot 1 (0x82)
`-T 0x82 -B 0 -b 7 -t 0x74`
- –System Node Controller 1 (0x72) on Cartridge Controller Slot 2 (0x84)
`-T 0x84 -B 0 -b 7 -t 0x72`

where:

- `-B <transit channel for bridged request (dual bridge)>`
- `-T <transit address for bridge request (dual bridge)>`
- `-b <ipmi channel number>`
- `-t <target slave address>`

Interfaces

IPMITool supports dynamic loading of interfaces that correspond to low-level communication methods for accessing IPMI systems. The most common of these are the System Interface provided by the OpenIPMI Linux kernel driver and IPMI over LAN interfaces.

System Interface

There are multiple types of system interfaces, and they are all similar enough to enable a single driver like OpenIPMI to support them all. They can be connected to any system bus such as ISA or X-bus that allows the main processor to access I/O mapped locations and meet the timing specifications. The varieties of system interfaces include KCS and SSIF. All of these are supported in recent versions of the OpenIPMI driver for the Linux kernel. IPMITool uses this driver to access the system interface through a character device node at `/dev/ipmi0`. To use this interface with IPMITool provide the `-l open` parameter on the command line.

LANPlus Interface

The LANPlus interface communicates with the MC over an ethernet LAN connection using UDP under IPv4. The LANPlus interface uses the RMCP+ protocol. RMCP+ facilitates improved authentication and data integrity checks as well as encryption and the ability to carry multiple types of payloads. Generic Serial Over LAN support requires RMCP+, so the IPMITool `sol activate` command requires the use of LANPlus.

RMCP+ session establishment uses a symmetric challenge-response protocol called RAKP (Remote Authenticated Key-Exchange Protocol) which allows the negotiation of many options.

NOTE: IPMITool does not allow the user to specify the value of every option, defaulting to the most obvious settings marked as required in the v2.0 specification. Authentication and integrity HMACS are produced with SHA1, and encryption is performed with AES-CBC-128. Role-level logins are not yet supported.

IPMITool must be linked with the OpenSSL library in order to perform the encryption functions and support the LANPlus interface. If the required packages are not found it will not be compiled and supported.

```
ipmitool -I lanplus -H <hostname> [-U <username>] [-P <password>] <command>
```

A hostname must be given on the command line in order to use the LAN interface with IPMITool.

The `-C` option allows the authentication integrity and encryption algorithms to be used for LANPlus sessions based on the cipher suite ID found in IPMI v2.0. The default cipher suite is 3 which specifies RAKP-HMAC-SHA1 authentication, HMAC-SHA1-96 integrity, and AES-CBC-128 encryption algorithms.

Example 1 Raw Get Device ID to chassis satellite controller over LAN

```
# ipmitool -I lanplus -H 16.85.178.125 -U admin -P admin123 -L Administrator -b 0 -t 0x44 raw 6 1  
15 01 02 01 02 29 0b 00 00 00 85 00 00 00 00
```

Example 2 Powering on C2N1 over LAN

```
# ipmitool -I lanplus -H 16.85.178.125 -U admin -P admin123 -L Administrator -B 0 -T 0x84 -b 7 -t 0x72 chassis  
power on
```

Chassis Power Control: Up/On

Example 3 Activating SOL on C2N1 over LAN

```
# ipmitool -I lanplus -H 16.85.178.125 -U admin -P admin123 -L Administrator -B 0 -T 0x84 -b 7 -t 0x72 sol  
activate
```

Activates SOL session for C2N1

Features

Instead of directly accessing the monitoring hardware for device entry, IPMI provides access to sensor data through abstracted messaging commands. Some common types of sensors that can be found in the system include baseboard and processor temperature sensors, processor and DIMM presence sensors, fan speed and failure monitoring, and baseboard, processor and SCSI terminating voltage sensors. The amount of data available for each sensor can be overwhelming, so by default IPMITool only displays the sensor name, reading and status. Considerably more output can be seen by enabling the verbose output option.

To facilitate discovery of features, IPMI includes a set of records called SDRs kept in a single centralized non-volatile storage area. These records include software information such as how many sensors are present, what type they are, their events, threshold info and more. This allows software to interpret and present sensor data without any prior knowledge about the platform.

Example 4 Output from `sdr list all` command

ZoMC	Static MC @ 20h	ok
254	Log FRU @FEh f0.60	ok
IPMB0 Phys Link	0x00	ok
ChasMgmtCtrlr1	Static MC @ 44h	ok
PsMgmtCtrlr1	Dynamic MC @ 52h	ok
PsMgmtCtrlr2	Dynamic MC @ 54h	ok
PsMgmtCtrlr3	Dynamic MC @ 56h	ok
PsMgmtCtrlr4	Dynamic MC @ 58h	ok
CaMC	Dynamic MC @ A6h	ok
CaMC	Dynamic MC @ A8h	ok
CaMC	Dynamic MC @ AAh	ok
CaMC	Dynamic MC @ ACh	ok
CaMC	Dynamic MC @ AEh	ok.
.		
.		
.		
CaMC	Dynamic MC @ A4h	ok

Example 5 Output from `sdr list all` command at cartridge

01-Front Ambient	21 degrees C	ok
02-CPU	40 degrees C	ok
03-DIMM	25 degrees C	ok
04-Cart Ctrlr	24 degrees C	ok
05-CPU Zone	29 degrees C	ok
06-LOM Zone	36 degrees C	ok
CaMC	Dynamic MC @ A4h	ok
SnMC	Dynamic MC @ 72h	ok
SnMC 1	Log FRU @01h c1.62	ok

See “[Verbose output examples](#)” (page 164) for an example of verbose output from the `sdr list all` command.

Events

Events are special messages sent by the management controller when they detect system management events. Some examples of events are temperature threshold exceeded, voltage threshold exceed, correctable ECC memory error, etc. These events are processed and usually logged in the SEL. This is similar to the SDR in that it provides a centralized non-volatile storage area for platform events that are logged autonomously by the MC or directly with event messages sent from the host.

There is an abundance of information available from an event log entry. By default IPMItool displays only the basic data for the event and the sensor that triggered it. Detailed information is available with the verbose option.

Example 6 Output from `sel list` command

0	04/16/2013	20:22:01	Power Supply #0x04	Failure detected	Asserted
1	06/28/2013	20:36:17	Power Supply #0x02	Presence detected	Deasserted
2	07/28/2013	00:20:52	Power Supply #0x02	Failure detected	Asserted
3	08/04/2013	00:23:10	Power Supply #0x02	Presence detected	Deasserted
4	08/09/2013	14:34:48	Fan #0x07	Transition to Off Line	Asserted
5	08/09/2013	14:34:49	Fan #0x07	Transition to Running	Deasserted

See “[Verbose output examples](#)” (page 164) for an example of verbose output from the `sel list` command.

Inventory

IPMI supports multiple sets of non-volatile FRU information for different parts in the system. This provides access to data such as serial number, part number, asset tag, and other information for major modules in the system including the baseboard, chassis, processors, memory, power supplies, and even the management controller itself. This information is even available when the system is powered down or non-operational, facilitating the creation of automated remote inventory and service applications. IPMITool can read and display full FRU information for the system as well as detailed descriptions of power supplies and full DIMM SPD data.

Example 7 Output from the `fru print` command

```
FRU Device Description : ChasMgmtCtrlr1
  Chassis Type          : Rack Mount Chassis
Chassis Part Number    : 700349-B21
Chassis Serial         : 600012J0SD
Chassis Extra          : d09701640003000000
Chassis Extra          : d1110102000000
Board Mfg Date         : Fri Dec 7 19:54:00 2012
Board Mfg              : HP
Board Product          : HP Moonshot 1500 Chassis Management Module
Board Serial           : 1G24900006J0SE
Board Part Number      : 712678-001
Board Extra            : d25835
Board Extra            : 700369-001
Product Manufacturer   : HP
Product Name           : HP Moonshot 1500 Chassis
Product Part Number    : 700451-001
```

Chassis management

This feature provides standardized chassis status and control functions that allow a remote system to be turned on/off or rebooted without manual intervention. It also provides commands for causing the chassis to physically identify itself with an implementation dependant mechanism such as turning on visible lights, displaying messages on an LCD, emitting beeps through a speaker, etc. IPMITool fully supports the available chassis management commands and can eliminate trips to the data center or server room to reset a frozen machine or help identify the single system in a rack that must be removed.

Example 8 Sample `chassis power` commands

```
root@JSMITH-LX:/# ipmitool -I lanplus -H ILOH101GEMINI -U Administrator -P password sdr list all
(output)
root@JSMITH-LX:/# ipmitool -I lanplus -H ILOH101GEMINI -U Administrator -P password -t 0xa4 power status
Chassis Power is on
```

In all of the above examples only a portion of the available output is shown, the full output is much richer and tells a full story about the system health and status; in addition verbose output options are available which increase the output information. See [“Verbose output examples” \(page 164\)](#) for examples of verbose output.

Synopsis

```
ipmitool [-chvV] [-Iopen <command>]
ipmitool [-chvV] -Ilan -H<hostname>
[-p<port>]
[-U<username>]
[-A<authtype>]
[-L<privlvl>]
[-aEPf<password>]
[-o<oemtype>]
```

```

<command>
ipmitool [-chvV] -Ilanplus -H<hostname>
[-p<port>]
[-U<username>]
[-L<privlvl>]
[-aEPf<password>]
[-o<oemtype>]
[-C<ciphersuite>]
<command>

```

Description

This program allows management of IPMI functions of either the local system via a kernel device driver or a remote system using IPMI v1.5 and IPMI v2.0. These functions include printing FRU information, LAN configuration, sensor readings and remove chassis power control.

IPMI management of a local system interface requires a compatible IPMI kernel driver to be installed and configured. On Linux this driver is called OpenIPMI and it is included in standard distributions.

Options

-a	Prompt for the remote server password.
-A <authtype>	Specify the authentication type to use during IPMI v1.5 LAN session activation. Supported types are NONE, PASSWORD, MD5 or OEM.
-c	Present output in CSV format. Not available with all commands.
-C<ciphersuite>	The remote server authentication, integrity, and encryption algorithms to use for IPMI v2 lanplus connections. Default = 3 and specifies RAKP-HMAC-SHA1 authentication, HMAC-SHA1-96 integrity, and AES-CBC-128 encryption algorithms.
-E	The remote server password is specified by the environment variable <code>ipmi_password</code> .
-f<password_file>	Specifies a file containing the remote server password. If this option is absent or if the <password_file> is empty the password defaults to NULL.
-h	Get basic usage help from the command line.
-H <address>	Remote server address can be IP address or hostname. This option is required for LAN and LANPLUS interfaces.
-I <interface>	Selects the IPMI interface. Supported interfaces display in the usage help output.
-L<privlvl>	Force session privilege level, defaults to admin.
-m<local address>	Set the local IPMB address. Default = 0x20.
-o<oemtype>	Select OEM type. Use <code>-o list</code> to see a list of currently supported OEM types.
-p<port>	Remote server UDP port. Default = 623.
-P<password>	Remote server password specified on the command line. It is not recommended to specify a password on the command line. NOTE: If no password method is specified, the IPMI tool prompts the user for a password, if no password is entered, the remote server password is set to NULL.
-t<target address>	Bridge IPMI requests to the remote target address.
-U<username>	Remote server username. Default = NULL

-v	Increase verbose output level. May be specified multiple times to increase levels of debug output, for example, specifying three times results in hexdumps of all incoming and outgoing packets.
-V	Display version information.

IPMITool Raw command syntax and example

1. Syntax — Target command towards specific virtual controller

- -b <ipmi channelnumber>
- -t <target slave address>
- -m <source slave address>
- Chassis controller -b 0 -t 0x44 -m 0x20
- Power supply A controller -b 0 -t 0x52 -m 0x20
- Power supply B controller -b 0 -t 0x54 -m 0x20
- Power supply C controller -b 0 -t 0x56 -m 0x20
- Power supply D controller -b 0 -t 0x58 -m 0x20

2. Examples:

- Raw Get Device ID to chassis satellite controller over LAN:

```
ipmitool -I lanplus -H 16.85.178.125 -U admin -P admin123 -L Administrator -b 0 -t 0x44 -m 0x20 raw 6 1
```
- Power on to C2N1 over LAN:

```
ipmitool -I lanplus -H 16.85.178.125 -U admin -P admin123 -L Administrator -B 0 -T 0x84 -b 7 -t 0x72 -m 0x20 chassis power on
```
- SOL to C2N1 over LAN

```
ipmitool -I lanplus -H 16.85.178.125 -U admin -P admin123 -L Administrator -B 0 -T 0x84 -b 7 -t 0x72 -m 0x20 sol activate
```


5 Command specification

IPMI provides standardized interfaces and commands for configuring the platform management subsystem. This enables cross-platform software to SDRs are an example of the interface for configuring sensor population and behavior on a system. There are also commands for configuring capabilities such as LAN and serial/modem remote protocols, user passwords and privilege levels, platform event filtering, alert destinations, and others.

This section provides specifications for elements that apply to all requests and responses.

See [“Completion codes” \(page 142\)](#).

Unless otherwise noted, reserved bits and fields in commands (request messages) and responses are written as 0. Applications must ignore the state of reserved bits when they are read.

Unless otherwise specified, commands that are listed as mandatory must be accessed via LUN 00b. An implementation may elect to make any command available on any LUN or channel as long as it does not conflict with other requirements in this specification.

Command table notation

The following section includes command tables that list the data that is included in a request or a response for each command. The completion code for a response is included as the first byte of the response data field for each command. The NetFn and command byte values for each command are specified in separate tables.

The following notation is used in the command tables.

Notation	Description
Request data	Identifies the portion of the table that lists the fields that are included in the data portion of a request message for the given command.
Response data	Identifies the portion of the table that lists the fields that are included in the data portion of a response message for the given command. The completion code is always listed as the first byte in the response data field.
4	Single byte field. A single value in the byte column of a command table is used to identify a single byte field. The value represents the offset to the field within the data portion of the message. In some cases a single byte field follows a variable length field in which case the single byte offset is represented with an alphabetic variable and number representing the single byte field's location relative to the end of the variable length field. For example: N+1.
5:7	Multi-byte field. The byte column indicates the byte offset(s) for a given field. For a multi-byte field, the first value indicates the starting offset, the second value (following the colon) indicates the offset for the last byte in the field. For example, 5:7 indicates a three-byte field spanning byte offsets 5, 6, and 7. In some cases, multi-byte fields may be variable length, in which case an alphabetic variable is used to represent the ending offset, for example: 5:N. Similarly, a field may follow a variable length field. In this case the starting value is shown as an offset relative to the notation used for the previous field, for example, if the previous field were 5:N, the next field would be shown starting at N+1. A variable length field may follow a variable length field, in which case a relative starting offset is shown with an alphabetic value indicating a relative ending offset, for example, N+1:M.
(3)	Optional Fields. When used in the byte column of the command tables, parentheses are used to indicate optional data byte fields. These can be absent or present at the choice of the party generating the request or response message. Devices receiving the message are required to accept any legal combination of optional data byte fields. Unless otherwise indicated, if an optional byte field is present, all prior specified byte fields must also be present. Similarly, if an optional byte field is absent all following byte fields must also be absent. For example, suppose a request accepts 4 data bytes. If data byte 3 was shown in parentheses as (3), it would indicate that byte 3 and following were optional. A legal request could consist of just bytes [1 and 2], bytes [1, 2, and 3,] or bytes [1, 2, 3 and 4]. A request which eliminates byte 3, but includes byte 4. (a request with data bytes [1, 2, and 4]), is illegal.

Notation	Description
	Multi-byte fields that are shown as optional cannot be split. Either all bytes for the field are present or absent. For example, if a four byte multi-byte field is listed as optional, it is illegal to include the first two bytes, but not the second two bytes.

Table 6 (page 26) lists the available Moonshot IPMI commands and, where available, the equivalent iLO Chassis Management CLI command. The IPMI commands and capabilities available for Moonshot are not completely analogous to the commands available at the iLO Chassis Management CLI, and so not every IPMI command has a CLI equivalent. Additionally, where there are analogous commands, the responses offered by the two commands may not be equivalent.

Table 6 Moonshot IPMI commands and their iLO CM CLI equivalents

Moonshot IPMI Command	NetFn	Command Code	Moonshot iLO CM CLI command equivalent
IPMI Device Global Commands			
Get Device ID	App (0x06)	0x01	Show chassis info provides a partially equivalent response.
Broadcast 'Get Device ID' ¹	App (0x06)	0x01	Show chassis info provides a partially equivalent response.
Cold Reset	App (0x06)	0x02	Reset CM
Warm Reset	App (0x06)	0x03	Reset CM
Get Self Test Results	App (0x06)	0x04	Show log IML provides a partially equivalent response (specifically, failures that were written to the iML)
Get ACPI Power State	App (0x06)	0x07	show node power CxNy
MC Watchdog Timer Commands			
Reset Watchdog Timer	App (0x06)	0x22	IPMI specific
Set Watchdog Timer	App (0x06)	0x24	IPMI specific
Get Watchdog Timer	App (0x06)	0x25	IPMI specific
IPMI Messaging Support Commands			
Set BMC Global Enables	App (0x06)	0x2E	IPMI specific
Get BMC Global Enables	App (0x06)	0x2F	IPMI specific
Clear Message Flags	App (0x06)	0x30	IPMI specific
Get Message Flags	App (0x06)	0x31	IPMI specific
Enable Message Channel Receive	App (0x06)	0x32	IPMI specific
Get Message	App (0x06)	0x33	IPMI specific
Send Message	App (0x06)	0x34	IPMI specific
Get System GUID	App (0x06)	0x37	The physical node UUID is not shown in the CLI.
Set System Info Parameters	App (0x06)	0x58	Not supported in CLI.
Get System Info Parameters	App (0x06)	0x59	Show node info returns the MAC address; other parameters returned by the IPMI command are IPMI specific.
Get Channel Authentication Capabilities	App (0x06)	0x38	IPMI specific

Table 6 Moonshot IPMI commands and their iLO CM CLI equivalents *(continued)*

Moonshot IPMI Command	NetFn	Command Code	Moonshot iLO CM CLI command equivalent
Set Session Privilege Level	App (0x06)	0x3B	IPMI specific
Close Session	App (0x06)	0x3C	IPMI specific
Get Session Info	App (0x06)	0x3D	IPMI specific
Get AuthCode	App (0x06)	0x3F	IPMI specific
Set Channel Access	App (0x06)	0x40	IPMI specific
Get Channel Access	App (0x06)	0x41	IPMI specific
Get Channel Info	App (0x06)	0x42	IPMI specific
Set User Access	App (0x06)	0x43	set user privilege
Get User Access	App (0x06)	0x44	show user
Set User Name	App (0x06)	0x45	add user
Get User Name	App (0x06)	0x46	show user
Set User Password	App (0x06)	0x47	set user password
Activate Payload	App (0x06)	0x48	IPMI specific
Deactivate Payload	App (0x06)	0x49	IPMI specific
Get Payload Activation Status	App (0x06)	0x4A	IPMI specific
Get Payload Instance Info	App (0x06)	0x4B	IPMI specific
Set User Payload Access	App (0x06)	0x4C	IPMI specific
Get User Payload Access	App (0x06)	0x4D	IPMI specific
Get Channel Payload Support	App (0x06)	0x4E	IPMI specific
Get Channel Payload Version	App (0x06)	0x4F	IPMI specific
Master Write-Read	App (0x06)	0x52	No CLI equivalent
Get Channel Cipher Suites	App (0x06)	0x54	IPMI specific
Suspend/Resume Payload Encryption	App (0x06)	0x55	IPMI specific
Set Channel Security Keys	App (0x06)	0x56	IPMI specific
Get System Interface Capabilities	App (0x06)	0x57	IPMI specific
Chassis Device Commands			
Get Chassis Capabilities	Chassis (0x00)	0x00	IPMI specific
Get Chassis Status	Chassis (0x00)	0x01	show chassis status or show node status
Chassis Control	Chassis (0x00)	0x02	set node power
Chassis Identify	Chassis (0x00)	0x04	set chassis uid or set cartridge uid
Set Power Restore Policy	Chassis (0x00)	0x06	set chassis autopower
Set System Boot Options	Chassis (0x00)	0x08	set node boot or set node bootonce
Get System Boot Options	Chassis (0x00)	0x09	show node boot
Get POH Counter	Chassis (0x00)	0x0F	No CLI equivalent

Table 6 Moonshot IPMI commands and their iLO CM CLI equivalents *(continued)*

Moonshot IPMI Command	NetFn	Command Code	Moonshot iLO CM CLI command equivalent
Event Commands			
Set Event Receiver	S/E (0x04)	0x00	IPMI specific
Get Event Receiver	S/E (0x04)	0x01	IPMI specific
Platform Event (Event Message)	S/E (0x04)	0x02	No CLI equivalent
Sensor Device Commands			
Get Device SDR Info	S/E (0x04)	0x20	IPMI specific
Get Device SDR	S/E (0x04)	0x21	show chassis status show chassis powersupply show chassis temperature show cartridge temperature
Reserve Device SDR Repository	S/E (0x04)	0x22	IPMI specific
Get Sensor Thresholds	S/E (0x04)	0x27	show chassis temperature and show cartridge temperature
Get Sensor Reading	S/E (0x04)	0x2D	show chassis status show chassis powersupply show chassis temperature show cartridge temperature
FRU Device Commands			
Get FRU Inventory Area Info	Storage (0x0A)	0x10	IPMI specific
Read FRU Data	Storage (0x0A)	0x11	show fru
Write FRU Data	Storage (0x0A)	0x12	No CLI equivalent
SDR Device Commands			
Get SDR Repository Info	Storage (0x0A)	0x20	IPMI specific
Get SDR Repository Allocation Info	Storage (0x0A)	0x21	IPMI specific
Reserve SDR Repository	Storage (0x0A)	0x22	IPMI specific
Get SDR	Storage (0x0A)	0x23	show chassis status show chassis powersupply show chassis temperature show cartridge temperature
Add SDR	Storage (0x0A)	0x24	IPMI specific
Delete SDR	Storage (0x0A)	0x26	IPMI specific
Clear SDR Repository	Storage (0x0A)	0x27	IPMI specific
Run Initialization Agent	Storage (0x0A)	0x2C	IPMI specific
SEL Device Commands			
Get SEL Info	Storage (0x0A)	0x40	IPMI specific
Reserve SEL	Storage (0x0A)	0x42	IPMI specific
Get SEL Entry	Storage (0x0A)	0x43	show log IML
Add SEL Entry	Storage (0x0A)	0x44	No CLI equivalent
Clear SEL	Storage (0x0A)	0x47	clear log

Table 6 Moonshot IPMI commands and their iLO CM CLI equivalents *(continued)*

Moonshot IPMI Command	NetFn	Command Code	Moonshot iLO CM CLI command equivalent
Get SEL Time	Storage (0x0A)	0x48	show time
Set SEL Time	Storage (0x0A)	0x49	set time
LAN Device Commands			
Set LAN Configuration Parameters	Transport (0x0C)	0x01	set network
Get LAN Configuration Parameters	Transport (0x0C)	0x02	show network
Serial/Modem Device Commands			
Set SOL Configuration Parameters	Transport (0x0C)	0x21	IPMI specific
Get SOL Configuration Parameters	Transport (0x0C)	0x22	IPMI specific
DCMI Specific commands			
Get DCMI Capability Info	DCGRP (0xDC)	0x01	IPMI specific
Get Asset Tag	DCGRP (0xDC)	0x06	show chassis asset
Get DCMI Sensor Info	DCGRP (0xDC)	0x07	show chassis temperature and show cartridge temperature
Set Asset Tag	DCGRP (0xDC)	0x08	set chassis asset
Get Controller Id String	DCGRP (0xDC)	0x09	IPMI specific
Set Controller Id String	DCGRP (0xDC)	0x0A	IPMI specific
PICMG Specific commands			
Get PICMG Properties	PICMG (0x00)	0x00	IPMI specific
Get Address Info	PICMG (0x00)	0x01	IPMI specific
Fru Inventory Device Lock Control	PICMG (0x00)	0x1F	IPMI specific

¹ Only relevant to satellite controllers.

NOTE: The IPMI commands and capabilities available for Moonshot are not completely analogous to those available to the iLO Chassis Management CLI, and so not every IPMI command has a CLI equivalent.

Standard command specification

This section presents the commands that are common to all IPMI devices that follow this specification's message/command interface. This includes management controllers that connect to the system via a compatible message interface, as well as IPMB devices.

Global commands

IPMI management controllers shall recognize and respond to these commands via LUN 0.

Get device ID command

This command is available to MC, ChMC, and PSMC.

This command is used to retrieve the intelligent device's hardware revision, firmware/software revision, and sensor and event interface command specification revision information. The command also returns information regarding the additional logical device functionality (beyond application and IPM device functionality) that is provided within the intelligent device, if any.

While broad dependence on OEM-specific functionality is discouraged, two fields in the response allow software to identify controllers for the purpose of recognizing controller specific functionality. These are the device ID and the product ID fields. A controller that just implements standard IPMI commands can set these fields to unspecified.

Table 7 Device ID command response data

Response data byte number	Data field
1	Completion code
2	Device ID. 00h = unspecified
3	Device revision <ul style="list-style-type: none"> • [7] — 1=device provides device SDRs and 0=device does not provide device SDRs. • [6:4] — Reserved. Return as 0. • [3:0] — Device revision, binary encoded.
4	Firmware revision 1 <ul style="list-style-type: none"> • [7] — Device available: 0=normal operation, device firmware, SDR repository update or self-initialization in progress. Firmware or SDR repository updates can be differentiated by issuing a <code>get SDR</code> command and checking the completion code. • [6:0] — Major firmware revision, binary encoded.
5	Firmware revision 2: minor firmware revision. BCD encoded.
6	IPMI version. Holds IPMI command specification version. BCD encoded. 00h = reserved. Bits 7:4 hold the least significant digit of the revision, while bits 3:0 hold the most significant bits. For example, a value of 51h indicates revision 1.5 functionality. 02h for implementations that provide IPMI v2.0 capabilities per this specification.
7	Additional device support (formerly called IPM device support) lists the IPMI logical device commands and functions supported by the controller that are in addition to the mandatory IPM and application commands. <ul style="list-style-type: none"> • [7] — Chassis device • [6] — Bridge (device responds to <code>bridge NetFn</code> commands) • [5] — IPMB event generator. Device generates event messages (platform event request messages) from the IPMB. • [4] — IPMB event receiver. Device accepts event messages (platform event request messages) from the IPMB. • [3] — FRU inventory device • [2] — SEL device • [1] — SDR repository device • [0] — Sensor device
8:10	manufacturer ID, LS byte first. The manufacturer ID is a 20-bit value that is derived from the IANA private enterprise ID (see below). Most significant four bits = reserved (0000b). 000000h = unspecified. 0FFFFFFh = reserved. This value is binary encoded. For example, the ID for the IPMI forum is 7154 decimal, which is 1BF2h, and would be stored in this record as F2h, 1Bh, 00h for bytes 8 through 10, respectively.
11:12	Product ID, LS byte first. This field can be used to provide a number that identifies a particular system, module, add-in card, or board set. The number is specified according to the manufacturer given by manufacturer ID (see below). 0000h = unspecified. FFFFh = reserved.
(13:16)	Auxiliary firmware revision information. This field is optional. If present, it holds additional information about the firmware revision, such as boot block or internal data structure version numbers. The meanings of the numbers are specific to the vendor identified by manufacturer ID (see below). When the

Table 7 Device ID command response data *(continued)*

Response data byte number	Data field
	vendor-specific definition is not known, generic utilities should display each byte as 2-digit hexadecimal numbers, with byte 13 displayed first as the most significant byte.

Additional specifications and descriptions for the device ID response fields:

Table 8 Additional device ID specifications

Device ID Specification	Description
Device ID/Device instance	<p>Specified by the manufacturer identified by the manufacturer ID field it allows controller-specific software to identify the unique application command, OEM fields, and functionality that are provided by the controller.</p> <p>Controllers that have different application commands, or different definitions of OEM fields, are expected to have different device ID values. Controllers that implement identical sets of applications commands can have the same device ID in a given system. Thus, a standardized controller could be produced where multiple instances of the controller are used in a system, and all have the same device ID value. (The controllers would still be differentiable by their address, location, and associated information for the controllers in the SDRs.)</p> <p>The device ID is typically used in combination with the product ID field such that the device IDs for different controllers are unique under a given product ID. A controller can optionally use the device ID as an instance identifier if more than one controller of that kind is used in the system.</p> <p>Binary encoded.</p>
Device revision	<p>The least significant nibble is used to identify when significant hardware changes have been made to the implementation of the management controller that cannot be covered with a single firmware release. This field is used to identify two builds off the same code firmware base, but for different board fab levels. For example, device revision "1" might be required for fab X and earlier boards, while device revision "2" would be for fab Y and later boards.</p> <p>Binary encoded and unsigned.</p>
Firmware revision 1	<p>Major revision of the firmware. 7-bits. It is incremented on major changes or extensions of the functionality of the firmware, such as additions, deletions, or modifications of the command set.</p> <p>The device available bit is used to indicate whether normal command set operation is available from the device, or if it is operating in a state where only a subset of the normal commands are available. Typically because the device is in a firmware update state. It may also indicate that full command functionality is not available because the device is in its initialization phase or an SDR update is in progress.</p> <p>The revision information obtained when the device available bit is 1 is indicative of the code version that is in effect. The version information may vary with the device available bit state.</p> <p>Binary encoded and unsigned.</p>
Firmware revision 2	<p>Minor revision of the firmware, incremented for minor changes such as bug fixes.</p> <p>BCD encoded.</p>
IPMI version	<p>The version of the IPMI specification in which the controller is compatible, indicating conformance with this document including event message formats and mandatory command support.</p> <p>The value is 02h for implementations that provide IPMI v2.0 capabilities per this specification.</p> <p>BCD encoded with bits 7:4 holding the least significant digit of the revision and bits 3:0 holding the most significant bits.</p>

Table 8 Additional device ID specifications *(continued)*

Device ID Specification	Description
Additional device support	Indicates the logical device support that the device provides in addition to the IPM and application logical devices.
Manufacturer ID	<p>Uses IANA (http://www.iana.org/). SMI network management private enterprise codes (enterprise numbers) for identifying the manufacturer responsible for the specification of functionality of the vendor (OEM) -specific commands, codes, and interfaces used in the controller.</p> <p>For example, an event in the SEL could have OEM values in the event record. An application that parses the SEL could extract the controller address from the event record contents and use it to send the <code>get device ID</code> command and retrieve the manufacturer ID. A manufacturer-specific application could then do further interpretation based on prior knowledge of the OEM field, while a generic cross-platform application would typically just use the ID to present the manufacturer's name alongside uninterpreted OEM event values.</p> <p>The manufacturer ID is for the manufacturer that defines the functionality of the controller, which is not necessarily the manufacturer that created the physical microcontroller. For example, vendor A may create the controller, but it gets loaded with vendor B's firmware. The manufacturer ID would be for vendor B, since they defined the controller's functionality.</p> <p>The manufacturer ID value from the <code>get device ID</code> command does not override manufacturer or OEM ID fields that are explicitly defined as part of a command or record format.</p> <p>If no vendor-specific functionality is defined, it is recommended that the field be loaded with the manufacturer ID for the manufacturer that is responsible for the controller's firmware, or the value FFFFh to indicate unspecified.</p> <p>Binary encoded and unsigned.</p>
Product ID	<p>Used in combination with the manufacturer ID and device ID values to identify the product-specific element of the controller-specific functionality. This number is specified by the manufacturer identified by the manufacturer ID field.</p> <p>Typically, a controller-specific application would use the product ID to identify the type of board, module, or system that the controller is used in, instead of using the data from the FRU information associated with the controller.</p>
Auxiliary firmware revision information	<p>This field is optional. If present, it holds additional information about the firmware revision, such as boot block or internal data structure version numbers. The meanings of the numbers are specific to the vendor identified by manufacturer ID. When the vendor-specific definition is not known, generic utilities should display each byte as 2-digit hexadecimal numbers, with byte 13 displayed first as the most significant byte.</p>

Cold reset command

This command is available to the MC. This command is not required to return a response in all implementations.

This command directs the responder's device to reinitialize its event, communication, and sensor functions. This causes the default setting of interrupt enables, event message generation, sensor scanning, threshold values, and other power up default state to be restored. If the device incorporates a self test, the self test also runs at this time.

Table 9 Cold reset command response data

Response data byte number	Data field
1	Completion code

NOTE: The `cold reset` command is provided for platform development, test, and platform-specific initialization and recovery actions. The system actions of the `cold reset` command are platform specific. Issuing a `cold reset` command could have adverse effects on system operation, particularly if issued during run-time. Therefore, the `cold reset` command should not be used unless all the side-effects for the given platform are known.

It is recognized that there are conditions where a given controller may not be able to return a response to a cold reset request message. Therefore, though recommended, the implementation is not required to return a response to the `cold reset` command. Applications should not rely on receiving a response as verification of the completion of a `cold reset` command.

Warm reset command

This command is available to the MC.

This command directs the responder's device to reset communications interfaces, but current configurations of interrupt enables, thresholds, and so on are left alone. A warm reset does not initiate the self test. The intent of the `warm reset` command is to provide a mechanism for cleaning up the internal state of the device and its communication interfaces. A `warm reset` resets communication state information such as sequence number and retry tracking, but does not reset interface configuration information such as addresses, enables, and so on. An application may try a warm reset if it determines a non-responsive communication interface, but it must also be capable of handling the side effects.

Table 10 Warm reset command response data

Response data byte number	Data field
1	Completion code

Get self test results command

This command is available to MC, ChMC, and PSMC.

This command directs the device to return its self test results, if any. A device implementing a self test normally runs that test on device power up as well as after `cold reset` commands. A device is allowed to update this field during operation if it has tests that run while the device is operating. Devices that do not implement a self test always return a 56h for this command.

While the self test only runs at particular times, the `get self test results` command can be issued any time the device is in a ready for commands state.

Table 11 Get self test results command response data

Response data byte number	Data field
1	Completion code.
2	<ul style="list-style-type: none"> 55h — No error. All self tests passed. 56h — Self test function not implemented in this controller. 57h — Corrupted or inaccessible data or devices. 58h — Fatal hardware error (system should consider MC inoperative). This indicates that the controller hardware (including associated devices such as sensor hardware or RAM) may need to be repaired or replaced. FFh — Reserved. All other — Device-specific internal failure. Refer to the particular device specification for definition.
3	<p>For byte 2 =:</p> <ul style="list-style-type: none"> 55h, 56h, FFh: 00h 58h, all other: Device-specific. 57h: self-test error bitfield. A return of 57h does not imply that all tests were run, just that a given test has failed. For example, 1b means failed, 0b means unknown. [7] — 1b = Cannot access SEL device. [6] — 1b = Cannot access SDR repository. [5] — 1b = Cannot access MC FRU device. [4] — 1b = IPMB signal lines do not respond. [3] — 1b = SDR repository empty. [2] — 1b = Internal use area of MC FRU corrupted. [1] — 1b = Controller update boot block firmware corrupted. [0] — 1b = Controller operational firmware corrupted.

Get ACPI power state command

This command is available to the MC.

The command is used to retrieve the present power state information that has been set into the controller. This is an independent setting from the system power state that may not necessarily match the actual power state of the system. Unspecified bits and codes are reserved and returned as 0.

Table 12 Get ACPI power state command response data

Response data byte number	Data field		
1	Completion code		
2	ACPI system power state		
	[7] — Reserved		
	[6:0] — System power state enumeration		
	00h	S0 / G0	Working
	01h	S1	Hardware context maintained, typically equates to processor/chip set clocks stopped
	02h	S2	Typically equates to stopped clocks with processor/cache context lost

Table 12 Get ACPI power state command response data *(continued)*

Response data byte number	Data field		
	03h	S3	Typically equates to suspend-to-RAM
	04h	S4	Typically equates to suspend-to-disk
	05h	S5 / G2	Soft off
	06h	S4/S5	Soft off, cannot differentiate between S4 and S5
	07h	G3	Mechanical off
	08h	sleeping	Sleeping - cannot differentiate between S1-S3
	09h	G1 sleeping	Sleeping - cannot differentiate between S1-S4
	0Ah	Override	S5 entered by override
	20h	Legacy on	Legacy on (indicates on for system that does not support ACPI or has ACPI capabilities disabled)
	21h	Legacy off	Legacy soft-off
	2Ah	Unknown	Power state has not been initialized, or device lost track of power state
3	ACPI device power state		
	[7] — Reserved		
	[6:0] — Device power state enumeration		
	00h	D0	
	01h	D1	
	02h	D2	
	03h	D3	
	02h	Unknown	Power state has not been initialized, or device lost track of power state

Broadcast get device ID command

This command is available to MC, ChMC, and PSMC. It is only relevant to satellite controllers.

This command is the broadcast version of the `get device ID` command which provides for the discovery of intelligent devices on the IPMB only. Request is formatted as an entire IPMB application request message, from the RsSA field through the second checksum, with the message prefixed with the broadcast slave address, 00h. Response format is same as the regular `get device ID` response.

The `broadcast get device ID` command is not bridged but can be delivered to the IPMB using master write-read commands.

To perform a discovery, the command is repeatedly broadcast with a different rsSA slave address parameter field specified in the command. The device that has the matching physical slave address information shall respond with the same data it would return from a regular (non-broadcast) `get device ID` command. Since an IPMB response message carries the responder's slave address, the response to the broadcast provides a positive confirmation that an intelligent device exists at the slave address given by the rsSA field in the request.

An application driving discovery then cycles through the possible range of IPMB device slave addresses to find the population of intelligent devices on the IPMB.

See “Get device ID command” (page 29) for information on the fields returned by the broadcast get device ID command response. The IPMB message format for the broadcast get device ID request exactly matches that for the get device ID command, with the exception that the IPMB message is prefixed with the 00h broadcast address. The following illustrates the format of the IPMB broadcast get device ID request message:

Figure 2 Broadcast get device ID request message

Broadcast (00h)	rsSA	netFn/rsLUN	check1	rqSA	rqSeq/rqLUN	Cmd (01h)	check2
--------------------	------	-------------	--------	------	-------------	--------------	--------

Addresses 00h-0Fh and F0h-FFh are reserved for I²C functions and not used for IPM devices on the IPMB. These addresses can therefore be skipped if using the broadcast get device ID command to scan for IPM devices. The remaining fields follow the regular IPMB definitions.

In order to speed the discovery process on the IPMB, a controller should drop off the bus as soon as it sees that the rsSA in the command does not match its rsSA.

IPMI messaging support commands

This section defines the commands used to support the system messaging interfaces. This includes control bits for using the MC as an event receiver and SEL device. SMM messaging and event message buffer support is optional. Use of IPMI support for SMI and SMM messaging is deprecated. Configuration interface support for enabling or disabling SMM messaging and corresponding SMI has been removed from the specification. If SMM messaging were implemented using the IPMI infrastructure, it would now be done as an OEM-proprietary capability.

System software that is not explicitly aware of the particular platform’s use of SMI messaging must assume that the any SMI options have been pre-configured by the controller, system BIOS, or other software. Therefore, runtime system software should not reconfigure SMI options, nor should it access the event message buffer if it finds that event message buffer interrupt is mapped to SMI. The effects of SMS accessing the event message buffer when it is configured for SMI are unspecified.

Set BMC global enables command

This command is available to the MC.

This command is used to enable message reception into message buffers, and any interrupt associated with that buffer getting full. The OEM0, OEM 1, and OEM 2 flags are available for definition by the OEM/system integrator. Generic system management software must not alter these bits.

Table 13 Set BMC global enables command request and response data

Request data byte number	Data field		
1	This field is set to xxxx_100xb on power-up and system resets. If the implementation allows the receive message queue interrupt to be enabled/disabled, the default for bit 0 should be 0b.		
	[7]	OEM 2 Enable	Generic system management software must do a ‘read-modifywrite’ using the get BMC global enables and set BMC global enables commands to avoid altering this bit.
	[6]	OEM 1 Enable	
	[5]	OEM 0 Enable	
	[4]	Reserved	
	[3]	1b =	Enable system event logging (enables/disables logging of events to the SEL - with the exception of events received over the system interface. Event reception and logging via the system interface is always enabled.) SEL logging is enabled by default whenever the MC is first powered up. It is recommended that this default state also be restored on system resets and power on.

Table 13 Set BMC global enables command request and response data *(continued)*

	[2]	1b =	Enable event message buffer. Error completion code returned if written as 1 and the event message buffer not supported.
	[1]	1b =	Enable event message buffer full interrupt.
	[0]	1b =	Enable receive message queue interrupt (this bit also controls whether KCS communication interrupts are enabled or disabled. An implementation is allowed to have this interrupt always enabled.)
	NOTE: If the event message buffer full or receive message queue interrupt are not supported, an implementation can elect to return a CCh error completion code for the <code>set BMC global enables</code> command if an attempt is made to enable the interrupt (this is the recommended implementation). Alternatively, the implementation can accept the command, but must return 0b for the corresponding bit in the <code>get BMC global enables</code> .		
Response data byte number	Data field		
1	Completion code		

Get BMC global enables command

This command is available to the MC.

This command is used to retrieve the present setting of the global enables. The OEM0, OEM 1, and OEM 2 flags are available for definition by the OEM/system integrator. Generic system management software must ignore these bits.

Table 14 Get BMC global enables command response data

Response data byte number	Data field		
1	Completion code		
2	[7]	1b =	OEM 2 enabled
	[6]	1b =	OEM 1 enabled
	[5]	1b =	OEM 0 enabled
	[4]	Reserved	
	[3]	1b =	System event logging enabled
	[2]	1b =	Event message buffer enabled
	[1]	1b =	Event message buffer full interrupt enabled
	[0]	1b =	Receive message queue interrupt enabled (this bit also indicates whether KCS communication interrupt is enabled or disabled.)
	If the receive message queue or event message full interrupts are not implemented the corresponding interrupt enabled status bit must return as 0b.		

Clear message flags command

This command is available to the MC.

This command is used to flush unread data from the receive message queue or event message buffer. This will also clear the associated buffer full/message available flags. See [“Get message flags command” \(page 38\)](#).

Table 15 Clear message flags command request and response data

Request data byte number	Data field		
1	[7]	1b =	Clear OEM 2
	[6]	1b =	Clear OEM 1
	[5]	1b =	Clear OEM 0
	[4]	Reserved	
	[3]	1b =	Clear watchdog pre-timeout interrupt flag
	[2]	Reserved	
	[1]	1b =	Clear event message buffer
	[0]	1b =	Clear receive message queue
	If the receive message queue or event message full interrupts are not implemented the corresponding interrupt enabled status bit must return as 0b.		
Response data byte number	Data field		
1	Completion code. Implementations are not required to return an error completion code if an attempt is made to clear the event message buffer flag but the event message buffer is not supported.		

Get message flags command

This command is available to the MC.

This command is used to retrieve the present message available states. The OEM0, OEM 1, and OEM 2 flags are available for definition by the OEM/system integrator. Generic system management software must ignore these bits.

Table 16 Get message flags command response data

Request data byte number	Data field		
1	Completion code		
2	Flags		
	[7]	1b =	OEM 2 data available.
	[6]	1b =	OEM 1 data available.
	[5]	1b =	OEM 0 data available.
	[4]	Reserved.	
	[3]	1b =	Watchdog pre-timeout interrupt occurred.
	[2]	Reserved.	
	[1]	1b =	Event message buffer full. Return as 0 if event message buffer is not supported, or when the event message buffer is disabled.
	[0]	1b =	Receive message available. One or more messages ready for reading from receive message queue.

Enable message channel receive command

This command is available to the MC.

This command is used to enable and disable message reception into the receive message queue from a given message channel. The command provides a mechanism to allow SMS to only receive

messages from channels that it intends to process, and provides a disable mechanism in case the receive message queue is being erroneously or maliciously flooded with requests on a particular channel. It does not affect the ability for SMS to transmit on that channel. Only the SMS message channel is enabled by default. All other channels must be explicitly enabled by BIOS or system software, as appropriate. It is recommended that a destination unavailable completion code be returned if a request message to SMS is rejected because reception has been disabled.

Table 17 Enable message channel receive command request and response data

Request data byte number	Data field
1	Channel number <ul style="list-style-type: none"> • [7:4] — Reserved • [3:0] — Channel number
2	Channel state <ul style="list-style-type: none"> • [7:2] — Reserved • [1:0] <ul style="list-style-type: none"> ◦ 00b = Disable channel ◦ 01b = Enable channel ◦ 10b = Gen channel enable/disable state ◦ 11b = Reserved
Response data byte number	Data field
1	Completion code
2	Channel number <ul style="list-style-type: none"> • [7:4] — Reserved • [3:0] — Channel number
3	Channel state <ul style="list-style-type: none"> • [7:1] — Reserved • [0] <ul style="list-style-type: none"> ◦ 1b = Channel enabled ◦ 0b = Channel disabled

Get message command

This command is available to the MC.

This command is used to get data from the receive message queue.

Software is responsible for reading all messages from the message queue even if the message is not the expected response to an earlier send message. System management software is responsible for matching responses up with requests.

The `get message` command includes an inferred privilege level that is returned with the message. This can help avoid the need for software to implement a separate privilege level and authentication mechanism. For example: A user activates a session with a maximum privilege level of Administrator on a multi-session channel, and an MD5 authentication type was negotiated. That user-level authentication is disabled. A user that has user or higher privilege can place messages into the receive message queue by sending them to LUN 10b, or by using the `send message` command. If the packet has authentication type = MD5, the packet is assigned an inferred privilege level

based the on the present operating privilege level for the user (set using the `set session privilege level` command). If, before sending the packet, the user had set their privilege level to Operator, the packet would be assigned an inferred privilege level of Operator. This means an authenticated (signed) packet can be assigned different inferred privilege levels based on the present operating privilege set by the `set session privilege level` command.) If the message is received in a packet that has authentication type = none, the packet is assigned an inferred privilege level of User, since that is the lowest privilege level for which that type of authentication is accepted.

Now suppose that the remote user had used the receive message queue as a way to send a message to system management software that requests a soft shutdown of the operating system. The message would either have an inferred privilege level of Operator or User depending on whether it was sent as an authenticated message or not. SMS can then use that inferred privilege level as part of deciding whether to accept and process the message or not. For single-session channels, the inferred privilege level is always set to the present operating privilege level. For session-less channels, the inferred privilege level is set to none, indicating that there was no IPMI-specified authentication operating on the channel from which the message was received.

Table 18 Get message command response data

Response data byte number	Data field
1	<p>Completion code.</p> <p>Generic, plus the command specific completion code: 80h = data not available (queue / buffer empty). Implementation of this completion code is mandatory. The code eliminates the need for system software to always check the message buffer flags to see if there is data left in the receive message queue. If a non-OK, non-80h completion is encountered - software must check the message flags to get the empty/non-empty status of the receive message queue.</p>
2	<p>Channel number</p> <ul style="list-style-type: none"> [7:4] Inferred privilege level for message. <ul style="list-style-type: none"> When the MC receives a message for the receive message queue, it assigns an inferred privilege level to the message as follows: <ul style="list-style-type: none"> If the message is received from a session-based channel, it is initially assigned a privilege level that matches the maximum requested privilege level that was negotiated via the <code>activate session</code> command. If per-message authentication is enabled, but user-level authentication is disabled, the MC assigns a level of User to any messages that are received with an authentication type = none. (Per-message and user-level authentication options only apply to multi-session channels). The MC then lowers the assigned privilege limit, if necessary, based on the present session privilege limit that was set via the <code>set session privilege level</code> command. If the channel is session-less such as IPMB), the MC returns none for the privilege level. 0h = None (unspecified) 1h = Callback level 2h = User level 3h = Operator level 4h = Administrator level 5h = OEM proprietary level [3:0] channel number
3:N	Message data. Maximum length and format dependent on protocol associated with the channel.

The following table indicates the contents of the Message Data field from the get message response according to the channel type and channel protocol that was used to place the message in the receive message queue.

Table 19 Get message data fields

	Originating channel type	Channel protocol	Message data for received requests (RQ) and responses (RS)
1	IPMB (I ² C)	IPMB ¹	RQ: netFn/rsLUN, chk1, rqSA, rqSeq/rqLUN, cmd, <data>, chk2 RS: netFn/rqLUN, chk1, rsSA, rqSeq/rsLUN, cmd, completion code, <data>, chk2
4	802.3 LAN	IPMB	RQ: Session handle, rsSWID, netFn/rsLUN, chk1, rqSWID or rqSA, rqSeq/rqLUN, cmd, <data>, chk2 RS: Session handle, rqSWID, netFn/rsLUN, chk1, rsSWID or rsSA, rqSeq/rsLUN, cmd, completion code, <data>, chk2
5	Asynch. serial/modem (RS-232)	IPMB (basic mode, terminal mode, and PPP mode)	RQ: Session handle, rsSWID, netFn/rsLUN, chk1, rqSWID or rqSA, rqSeq/rqLUN, cmd, <data>, chk2 RS: Session handle, rqSWID, netFn/rsLUN, chk1, rsSWID or rsSA, rqSeq/rsLUN, cmd, completion code, <data>, chk2 NOTE: When LUN 10b is used to deliver a message to SMS from a terminal mode remote console, the MC inserts fixed values for the SWIDs and LUNs in the message. Messages from the remote console are always returned as coming from SWID 40h (81h) LUN 00b, and going to SMS SWID 20h (41h) LUN 00b.
6	Other LAN	IPMB	RQ: Session handle, rsSWID, netFn/rsLUN, chk1, rqSWID or rqSA, rqSeq/rqLUN, cmd, <data>, chk2 RS: Session handle, rqSWID, netFn/rsLUN, chk1, rsSWID or rsSA, rqSeq/rsLUN, cmd, completion code, <data>, chk2
7	PCI SMBus	IPMI-SMBus	RQ: rsSA, Netfn(even)/rsLUN, 00h, rqSA, rqSeq/rqLUN, CMD, <data>, PEC RS: rqSA or rqSWID, NetFn(odd)/rqLUN, 00h, rsSA or rsSWID, rqSeq/rsLUN, CMD, completion code, <data>, PEC
8	SMBus v1.0/1.1		
9	SMBus v2.0		
10	Reserved for USB 1.x	n/a	n/a
11	Reserved for USB 2.x	n/a	n/a
12	System interface	BT, KCS, SMIC	RQ: Session handle, rsSWID, netFn/rsLUN, chk1, rqSWID or rqSA, rqSeq/rqLUN, cmd, <data>, chk2 RS: Session handle, rqSWID, netFn/rsLUN, chk1, rsSWID or rsSA, rqSeq/rsLUN, cmd, completion code, <data>, chk2

¹ This message data matches the IPMB message format with the leading slave address omitted. The format includes checksums. In order to verify those checksums, they must be calculated as if 20h (MC slave address) was the value that was used as the slave address when the checksums were calculated per [IPMB]. 20h is always used for the checksum

calculation for the receive message queue data whenever IPMB is listed as the originating bus and with IPMB as the channel protocol.

Send message command

This command is available to the MC.

The `send message` command is used for bridging IPMI messages between channels, and between the SMS and a given channel.

For IPMI v2.0 the `send message` command has been updated to include the ability to indicate whether a message must be sent authenticated or with encryption (for target channels on which authentication and/or encryption are supported and configured).

Table 20 Send message command request and response data

Request data byte number	Data field		
1	Channel number		
	[7:6]	00b =	No tracking. The MC reformats the message for the selected channel but does not track the originating channel, sequence number, or address information. This option is typically used when software sends a message from the system interface to another media. Software will typically use no tracking when it delivers sends a message from the system interface to another channel, such as IPMB. In this case, the software formats the encapsulated message so that when it appears on the other channel, it appears to have been directly originated by MC LUN 10b. See “MC LUN 10b” (page 148) for more information.
		01b =	Track request. The MC records the originating channel, sequence number, and addressing information for the requester, and then reformats the message for the protocol of the destination channel. When a response is returned, the MC looks up the requester’s information and format the response message with the framing and destination address information and reformats the response for delivery back to the requester. This option is used for delivering IPMI request messages from non-SMS (non-system interface) channels. See “Send Message command with response tracking” (page 149) for more information.
		10b =	Send raw (optional). This option is primarily provided for test purposes. It may also be used for proprietary messaging purposes. The MC simply delivers the encapsulated data to the selected channel in place of the IPMI message data. If the channel uses sessions, the first byte of the message data field must be a session handle. The MC must return a non-zero completion code if an attempt is made to use this option for a given channel and the option is not supported. It is recommended that completion code CCh be returned for this condition.
		11b =	Reserved
	[5]	1b =	Send message with encryption. MC returns an error completion code if this encryption is unavailable.
		0b =	Encryption not required. The message is sent unencrypted if that option is available under the given session. Otherwise, the message is sent encrypted.
	[4]	1b =	Send message with authentication. MC returns an error completion code if this authentication is unavailable.
		0b =	Authentication not required. Behavior is dependent on whether authentication is used and whether the target channel is running an IPMI v1.5 or IPMI v2.0/RMCP+ session, as follows: <ul style="list-style-type: none"> IPMI v1.5 sessions default to sending the message with authentication if that option is available for the session. IPMI v2.0/RMCP+ sessions send the message unauthenticated if that option is available under the session. Otherwise, the message is sent with authentication.

Table 20 Send message command request and response data *(continued)*

	[3:0]	Channel number where to send the message
2:N	Message data. Format dependent on target channel type.	
Request data byte number	Data field	
1	<p>Completion code</p> <p>Generic, plus additional command-specific completion codes: 80h = Invalid session handle. The session handle does not match up with any currently active sessions for this channel.</p> <p>If channel medium = IPMB, SMBus, or PCI management bus (This status is recommended for applications that need to access low-level I²C or SMBus devices).</p> <ul style="list-style-type: none"> • 81h = Lost arbitration • 82h = Bus error • 83h = NAK on write 	
(2:N)	<p>Response data</p> <p>This data will only be present when using the <code>send_message</code> command to originate requests from IPMB or PCI management bus to other channels such as LAN or serial/modem. It is not present in the response to a <code>send_message</code> command delivered via the system interface.</p>	

NOTE: The MC does not parse messages that are encapsulated in a `send_message` command and does not know what privilege level should be associated with an encapsulated message. Thus, messages that are sent to a session using the `send_message` command are always output using the authentication type that was negotiated when the session was activated.

The following table summarizes the contents of the message data field when the `send_message` command is used to deliver an IPMI message to different channel types. In most cases, the format of message information the message data field follows are the same used for the IPMB, with two typical exceptions:

- When the message is delivered to channels without physical slave devices, a SWID field takes the place of the slave address field.
- When the message is delivered to a channel that supports sessions, the first byte of the message data holds a session handle.

Table 21 Send message data fields

	Target channel type	Target channel protocol	Message data for sending requests (RQ) and responses (RS)
1	IPMB (I ² C)	IPMB	<p>RQ: rsSA, netFn/rsLUN, chk1, rqSA, rqSeq/rqLUN, cmd, <data>, chk2</p> <p>RS: rqSA, netFn/rqLUN, chk1, rsSA, rqSeq/rsLUN, cmd, completion code, <data>, chk2</p>
4	802.3 LAN	IPMB+session header	<p>RQ: Session handle¹, rsSWID, netFn/rsLUN, chk1, rqSWID or rqSA, rqSeq/rqLUN, cmd, <data>, chk2</p> <p>RS: Session handle¹, rqSWID, netFn/rsLUN, chk1, rsSWID or rsSA, rqSeq/rsLUN, cmd, completion code, <data>, chk2</p>
5	Asynch. serial/modem (RS-232)	IPMB (basic mode, terminal mode, and PPP mode)	<p>RQ: Session handle¹, rsSWID, netFn/rsLUN, chk1, rqSWID or rqSA, rqSeq/rqLUN, cmd, <data>, chk2</p> <p>RS: Session handle¹, rqSWID, netFn/rsLUN, chk1, rsSWID or rsSA, rqSeq/rsLUN, cmd, completion code, <data>, chk2</p>

Table 21 Send message data fields *(continued)*

	Target channel type	Target channel protocol	Message data for sending requests (RQ) and responses (RS)
			NOTE: Terminal mode has a single, fixed SWID for the remote console. Software using send message to deliver a message to a terminal mode remote console should use their SWID or slave address as the source of the request or response, and the terminal mode SWID (40h) as the destination.
6	Other LAN	IPMB	RQ: Session handle ¹ , rsSWID, netFn/rsLUN, chk1, rqSWID or rqSA, rqSeq/rqLUN, cmd, <data>, chk2 RS: Session handle ¹ , rqSWID, netFn/rsLUN, chk1, rsSWID or rsSA, rqSeq/rsLUN, cmd, completion code, <data>, chk2
7	PCI SMBus	IPMI-SMBus	RQ: rsSA, netFn/rsLUN, chk1, rqSA, rqSeq/rqLUN, cmd, <data>, chk2
8	SMBus v1.0/1.1		RS: rqSA, netFn/rqLUN, chk1, rsSA, rqSeq/rsLUN, cmd, completion code, <data>, chk2
9	SMBus v2.0		
10	Reserved for USB 1.x	n/a	n/a
11	Reserved for USB 2.x	n/a	n/a
12	System interface		RQ: rsSA, netFn/rsLUN, chk1, rqSA, rqSeq/rqLUN, cmd, <data>, chk2 RS: rqSA, netFn/rqLUN, chk1, rsSA, rqSeq/rsLUN, cmd, completion code, <data>, chk2 NOTE: MC adds session handle information when it puts the message into the receive message queue .

¹ The session handle identifies a particular active session on the given channel. The MC assigns a different value to each time a new session is activated. A typical implementation keeps track of the last value that was assigned and increment it before assigning it to a new active session when the `activate session` command has been accepted. Software must include this field for channels where the `get channel info` command indicates that the channel supports sessions.

Get system GUID command

This command is available to the MC.

This optional, though highly recommended, command can be used to return a GUID (also known as a UUID), for the managed system to support the remote discovery process and other operations. The format of the ID follows the octet format specified in [RFC 4122]. [RFC4122] specifies four different versions of UUID formats and generation algorithms suitable for use for a GUID in IPMI.

- Time based — version 1 (0001b)
- Name based:
 - version 3 (0011b) MD5 hash
 - version 4 (0100b) Pseudo-random
 - version 5 (SHA1 hash)

[SMBIOS] does not specify a particular specification or version for UUID generation. In general, if it remains unspecified, the version 1 format is recommended by the IPMI specification for new

system implementations. However, versions 3, 4, or 5 formats are also allowed. A system GUID should not change over the lifetime of the system.

If the MC is on a removable card that can be moved to another system, the vendor of the card or system vendor should provide a mechanism for generating a new system GUID or retrieving the SMBIOS UUID from the given system.

Since the GUID is typically permanently assigned to a system, an interface that would allow the GUID to be configured or changed is not specified. For systems that support [SMBIOS], the system GUID that is returned by the MC should match the UUID field value in the SMBIOS system information (type 1) record.

The session header (session request data and session response data) values shown in the following table illustrate the values that would be used to execute a `get system GUID` command outside of an active session. The `get system GUID` is always accepted outside of an active session. The `get system GUID` command can also be executed within the context of an active session (providing the user is operating at higher than callback privilege). When the `get system GUID` command is executed in the context of an active session, the session header fields must have correct values according to the authentication, session ID, and session sequence number information that was negotiated for the session.

Session header fields request and response data prior when used prior to session activation

authentication type = NONE

session seq# = null (0's)

Session ID = null (0's)

AuthCode = NOT PRESENT

Table 22 Get system GUID command response data

Response data byte number	Data field
1	Completion code
2:17	GUID bytes 1 through 16.

Set system info parameters command

This command is available to the MC.

This command is used for setting system information parameters such as system name and BIOS/system firmware revision information.

Table 23 Set system info parameters command request and response data

Request data byte number	Data field
1	Parameter selector
2:N	Configuration parameter data, per Table 25 (page 46)
Response data byte number	Data field
1	Completion code <ul style="list-style-type: none">80h = parameter not supported81h = attempt to set the set in progress value (in parameter #0) when not in the set complete state. (This completion code provides a way to recognize that another party has already claimed the parameters).82h = attempt to write read-only parameter

Get system info parameters command

This command is available to the MC.

This command is used for retrieving system information parameters from the set system info parameters command.

Table 24 Get system info parameters command request and response data

Request data byte number	Data field
1	[7] <ul style="list-style-type: none"> 0b = get parameter 1b = get parameter revision only [6:0] — reserved
2	Parameter selector
3	Set selector. Selects a given set of parameters under a given parameter selector value. 00h if parameter does not use a set selector.
4	Block selector (00h if parameter does not require a block number)
Response data byte number	Data field
1	Completion code Generic codes, plus the command-specific completion code: 80h = parameter not supported.
2	[7:0] — Parameter revision. Format: <ul style="list-style-type: none"> MSN = present revision LSN = oldest revision parameter that is backward compatible 11h for parameters in this specification
The following data bytes are not returned when the get parameter revision only bit is 1b.	
3:N	Configuration parameter data, per “ System info parameters ” (page 46). If the rollback feature is implemented, the MC makes a copy of the existing parameters when the set in progress state becomes asserted. (See the set in progress parameter #0). While the set in progress state is active, the MC returns data from this copy of the parameters, plus any uncommitted changes that were made to the data. Otherwise, the MC returns parameter data from non-volatile storage.

Table 25 System info parameters

Parameter	#	Parameter data (non-volatile unless otherwise noted) ¹	
Set in progress (volatile)	0	Data 1 - This parameter is used to indicate when any of the following parameters are being updated, and when the updates are completed. The bit is primarily provided to alert software that some other software or utility is in the process of making changes to the data. An implementation can also elect to provide a rollback feature that uses this information to decide whether to roll back to the previous configuration information, or to accept the configuration change. If used, the roll back restores all parameters to their previous state. Otherwise, the change takes effect when the write occurs.	
		[7:2]	Reserved
		[1:0]	00b = Set complete. If a system reset or transition to powered down state occurs while set in progress is active, the MC goes to the set complete state. If rollback is implemented, going directly to set complete without first doing a commit write causes any pending write data to be discarded.

Table 25 System info parameters *(continued)*

Parameter	#	Parameter data (non-volatile unless otherwise noted) ¹	
		01b =	Set in progress indicating that some utility or other software is presently doing writes to parameter data. It is a notification flag only, it is not a resource lock. The MC does not provide any interlock mechanism that would prevent other software from writing parameter data while set in progress value is present on these bits.
		10b =	Commit write (optional). This is only used if a rollback is implemented. The MC saves the data that has been written since the last time the set in progress and then go to the set in progress state. An error completion code is returned if this option is not supported.
		11b =	Reserved
System firmware version	1	System firmware version string in text. System firmware that requires multiple strings to represent version information can separate those strings using 00h as the delimiter for ASCII+LATIN1 and UTF-8 encoded string data, or 0000h for UNICODE encoded string data. For IA32 and EMT64 utilizing non-EFI system firmware, it is recommended that four blocks (64 bytes) of storage be provided. For EFI-based systems, 256 bytes is recommended.	
		<p>NOTE: System firmware may optionally include a routine that checks during POST to see if this parameter is up-to-date with the present firmware version, and if not, update it automatically. Other implementations may elect to automatically update this parameter when system firmware updates occur.</p>	
		Data 1 —	set selector = 16-byte data block number to access, 0 based. Two data blocks (32-bytes) for string data required, at least three recommended. Number of effective characters is dependent on the encoding selected in string data byte 1.
		Data 2:17 —	<p>16-byte block for system firmware name string data</p> <p>For the first block of string data (set selector = 0), the first two bytes indicate the encoding of the string and its overall length as follows:</p> <p>String data byte 1:</p> <ul style="list-style-type: none"> [7:4] — Reserved [3:0] — Encoding <ul style="list-style-type: none"> 0h = ASCII+Latin1 1h = UTF-8 2h = UNICODE All other = Reserved <p>String data byte 2:</p> <ul style="list-style-type: none"> [7:0] - String length (in bytes, 1-based)
		Data 1 —	set selector = 16-byte data block number to access, 0 based. Two data blocks (32-bytes) for string data required, at least three recommended. Number of effective characters will be dependent on the encoding selected in string data byte 1.
		Data 2:17 -	<p>16-byte block for system name string data</p> <p>For the first block of string data (set selector = 0), the first two string data bytes indicate the encoding of the string and its overall length</p>
System name	2	System name. A name for the overall system to be associated with the MC. This may or may not match other names that are used for the system.	
		Data 1 —	set selector = 16-byte data block number to access, 0 based. Two data blocks (32-bytes) for string data required, at least three recommended. Number of effective characters will be dependent on the encoding selected in string data byte 1.
		Data 2:17 -	<p>16-byte block for system name string data</p> <p>For the first block of string data (set selector = 0), the first two string data bytes indicate the encoding of the string and its overall length</p>

Table 25 System info parameters (continued)

Parameter	#	Parameter data (non-volatile unless otherwise noted) ¹	
			<p>as follows. There is no required value to be set or used for any bytes that are past the string length.</p> <p>String data byte 1:</p> <ul style="list-style-type: none"> • [7:4] — Reserved • [3:0] — Encoding <ul style="list-style-type: none"> ◦ 0h = ASCII+Latin1 ◦ 1h = UTF-8 ◦ 2h = UNICODE ◦ All other = Reserved <p>String data byte 2:</p> <ul style="list-style-type: none"> • [7:0] - String length (in bytes, 1-based)
Primary operating system name (non-volatile)	3		<p>Primary operating system name. The OS that the system boots to for this MC according to the default configuration of its system firmware. In systems that may have multiple physical partitions, this reflects the OS for the partition that holds the given MC. For systems that have virtual machine capability being utilized (where more than one virtual systems may be sharing a physical MC), it is recommended that this value hold the name of the virtual machine monitor (VMM) software or VMM type).</p>
		Data 1	<p>Set selector = 16-byte data block number to access, 0 based. Two data blocks (32-bytes) for string data required, at least three recommended. Number of effective characters will be dependent on the encoding selected in string data byte 1.</p>
		Data 2:17	<p>16-byte block for system name string data</p> <p>For the first block of string data (set selector = 0), the first two bytes indicate the encoding of the string and its overall length as follows. There is no required value to be set or used for any bytes that are past the string length.</p> <p>String data byte 1:</p> <ul style="list-style-type: none"> • [7:4] — Reserved • [3:0] — Encoding <ul style="list-style-type: none"> ◦ 0h = ASCII+Latin1 ◦ 1h = UTF-8 (ls-byte first) ◦ 2h = UNICODE (ls-byte first) ◦ All other = Reserved <p>String data byte 2:</p> <ul style="list-style-type: none"> • [7:0] - string length (in bytes, 1-based)
Operating system name (volatile)	4		<p>Present operating system name. The name of the operating system that is presently running and able to access this MC's system interface. The MC automatically clears this value by zeroing out the string length on system power cycles and resets.</p> <p>In systems that may have multiple physical partitions, this reflects the OS for the partition that holds the given MC. For systems that have virtual machine capability being utilized (where more than one virtual systems may be sharing a physical MC), it is recommended that this value hold the name of the virtual machine monitor (VMM) software or VMM type.</p>

Table 25 System info parameters *(continued)*

Parameter	#	Parameter data (non-volatile unless otherwise noted) ¹	
		Data 1	Set selector = 16-byte data block number to access, 0 based. Two data blocks (32-bytes) for string data required, at least three recommended. Number of effective characters is dependent on the encoding selected in string data byte 1.
		Data 2:17	16-byte block for system name string data For the first block of string data (set selector = 0), the first two bytes indicate the encoding of the string and its overall length as follows. There is no required value to be set or used for any bytes that are past the string length. String data byte 1: <ul style="list-style-type: none"> • [7:4] — Reserved • [3:0] — Encoding <ul style="list-style-type: none"> ◦ 0h = ASCII+Latin1 ◦ 1h = UTF-8 ◦ 2h = UNICODE ◦ All other = Reserved String data byte 2: <ul style="list-style-type: none"> • [7:0] - String length (in bytes, 1-based)
OEM	192 ... 255	This range is available for special OEM system information parameters.	

¹ Choice of system manufacturing defaults for non-volatile parameters is left to the system manufacturer unless otherwise specified.

Master write-read command

This command can be used for low-level I²C/SMBus write, read, or write-read access to the IPMB or private busses behind a management controller. The command can also be used for providing low-level access to devices that provide an SMBus slave interface.

NOTE: In Moonshot, this command is not available over LAN.

Table 26 Master write-read command request and response data

IPMI request data byte number	Data field		
1	Bus ID:		
	[7:4]	Channel number (ignored when bus type=1b)	
	[3:1]	Bus ID, 0-based (always 000b for public bus ([bus type=0b]))	
	[0]	Bus type:	
		0b =	Public (for example, IPMB or PIC Management) bus. The channel number value is used to select the target bus.
2		1b =	Private bus (the bus ID value is used to select the target bus.)
	Requested maximum privilege level		
	[7:1]	Slave address	
	[0]	Reserved. Write a 0.	

Table 26 Master write-read command request and response data (continued)

3	Read count. Number of bytes to read, 1 based. 0 equals not bytes to read. <i>The maximum read count should be at least 34 bytes.</i> This allows the command to be used for an SMBus Block Read. This is required if the command provides access to an SMBus or IPMB. Otherwise, if FRU SEEPROM devices are accessible, at least 31 bytes must be supported. Note that an implementation that supports fewer bytes can be supported if none of the devices to be accessed can handle the recommended minimum.
4:N	Data to write. This command should support <i>at least 35 bytes</i> of write data. This allows the command to be used for an SMBus Block Write with PEC. Otherwise, if FRU SEEPROM devices are accessible, at least 31 bytes must be supported. Note that an implementation is allowed to support fewer bytes if none of the devices to can handle the recommended minimum.
IPMI response data byte number	Data field
1	Completion code A management controller shall return an error Completion Code if an attempt is made to access an unsupported bus. This is a generic response but also may include the following command specific codes: <ul style="list-style-type: none"> • 81h—Lost arbitration • 82h—Bus error • 83h—NAK on write • 84h—Truncated read
(2:M)	Bytes read from specified slave address. This field will be absent if the read count is 0. The controller terminates the I ² C transaction with a STOP condition after reading the requested number of bytes.

Get channel authentication capabilities command

This command is available to the MC.

This command is sent in unauthenticated (clear) format. This command is used to retrieve capability information about the channel from which the message is delivered, or for a particular channel. The command returns the authentication algorithm support for the given privilege level. When activating a session, the privilege level passed in this command is normally the same requested maximum privilege level that is used for a subsequent `activate session` command.

MC implementations of IP-based channels must support the `get channel authentication capabilities` command using the IPMI v1.5 packet format. BMCs that support IPMI v2.0 RMCP+ must also support the command using the IPMI v2.0/RMCP+ format.

The `get channel authentication capabilities` command can also be used as a no-op “ping” to keep a session from timing out.

Session header fields request and response data prior when used prior to session activation

authentication type = NONE/payload type = IPMI message

session seq# = null (0's)

Session ID = null (0's)

AuthCode = NOT PRESENT

Table 27 Get channel authentication capabilities command request and response data

IPMI request data byte number	Data field
1	Channel number

Table 27 Get channel authentication capabilities command request and response data *(continued)*

	[7]	1b =	Get IPMI v2.0+ extended data. If the given channel supports authentication but does not support RMCP+ (such as a serial channel), then the response data should return with bit [5] of byte 4 = 0b, byte 5 should return 01h,
		0b =	Backward compatible with IPMI v1.5. Result response data only returns bytes 1:9, bit [7] of byte 3 (authentication type support) and bit [5] of byte 4 returns as 0b, bit [5] of byte 5 returns 00h.
	[6:4]	Reserved	
	[3:0]	Channel number	
		0hBh, Fh =	Channel numbers
		Eh =	Retrieve information for the channel from which this request was issued.
2	Requested maximum privilege level		
	[7:4]	Reserved	
	[3:0]	Requested privilege level	
		0h =	Reserved
		1h =	Callback level
		2h =	User level
		3h =	Operator level
		4h =	Administrator level
		5h =	OEM proprietary level
IPMI response data byte number	Data field		
1	Completion code		
2	Channel number on which the authentication capabilities is being returned. If the channel number in the request was set to Eh, this returns the channel number for the channel where the request was received.		
3	Authentication type support. Returns the setting of the authentication type enable field from the configuration parameters for the given channel that corresponds to the requested maximum privilege level.		
	[7]	1b =	IPMI v2.0+ extended capabilities available. See Extended capabilities field below.
		0b =	IPMI v1.5 support only.
	[6]	Reserved	
	[5:0]	IPMI v1.5 authentication type(s) enabled for given requested maximum privilege level.	
		All bits:	1b = Supported 0b = Authentication type not available for use
		[5]	OEM proprietary (per OEM identified by the IANA OEM ID in the RMCP ping response)
		[4]	Straight password / key
		[3]	Reserved
		[2]	MD5
		[1]	MD2

Table 27 Get channel authentication capabilities command request and response data (continued)

		[0]	None
4	[7:6]	Reserved	
	[5]	KG status (two-key login status). Applies to v2.0/RMCP+ RAKP authentication only. Otherwise, ignore as reserved.	
		0b =	KG is set to default (all 0's). User key KUID is used in place of KG in RAKP. (Knowledge of KG not required for activating session.)
		1b =	KG is set to non-zero value. (Knowledge of both KG and user password (if not anonymous login) required for activating session.)
	Following bits apply to IPMI v1.5 and v2.0:		
	[4]	Per-message authentication status	
		0b =	Per-message authentication is enabled. Packets to the MC must be authenticated per authentication type used to activate the session, and the user level authentication setting.
		1b =	Per-message authentication is disabled. Authentication type none accepted for packets to the MC after the session has been activated.
	[3]	User level authentication status	
		0b =	User level authentication is enabled. User level commands must be authenticated per authentication type used to activate the session.
		1b =	User level authentication is disabled. Authentication type none accepted for user level commands to the MC.
	[2:0]	Anonymous login status. This parameter returns values that tells the remote console whether there are users on the system that have 'null' usernames. This can be used to guide the way the remote console presents login options to the user.	
		[2]	1b = Non-null usernames enabled. (One or more users are enabled that have non-null usernames).
		[1]	1b = Null usernames enabled. (One or more users that have a null username, but non-null password, are presently enabled).
		[0]	1b = Anonymous login enabled. (A user that has a null username and null password is presently enabled).
5	For IPMI v1.5: - Reserved		
	For IPMI v2.0+: - Extended capabilities		
	[7:2]	Reserved	
	[1]	1b =	Channel supports IPMI v2.0 connections
	[0]	1b =	Channel supports IPMI v1.5 connections
6:8	OEM ID. IANA enterprise number for OEM/Organization that specified the particular OEM authentication type for RMCP. Least significant byte first. Return 00h, 00h, 00h if no OEM authentication type available.		
9	OEM auxiliary data. Additional OEM-specific information for the OEM authentication type for RMCP. Return 00h if no OEM authentication type available.		

Get Channel Cipher Suites Command

This command can be executed prior to establishing a session with the MC. The command is used to look up what authentication, integrity, and confidentiality algorithms are supported. The algorithms are used in combination as 'Cipher Suites'. This command only applies to implementations that support IPMI v2.0/RMCP+ sessions.

The data is accessed 16-bytes at a time starting from List Index field value of 0 in the request and then repeating the request incrementing the List Index field each time until fewer than 16-bytes of algorithm data (or no algorithm data) is returned in the response, or the maximum List Index value has been reached.

A given Cipher Suite may only be available for establishing a session at a particular maximum privilege level or lower. For example, a Cipher Suite that has a privilege level of 'Admin' can therefore be used for any privilege level, while a privilege level of User can only be used for establish sessions with a Maximum Requested Privilege Level of User or Callback.

Because the authentication algorithm specifies the steps for authenticating the user, it is a necessary part of session establishment. Therefore an authentication algorithm number is required for all Cipher Suites. It is possible that a given Cipher Suite may not specify use of an integrity or confidentiality algorithm. If the Cipher Suite has integrity and/or confidentiality of 'none', then all the same steps for establishing a session are used (open session request/response, RAKP messages) - but the integrity (AuthCode) and confidentiality fields will be absent in packets for that are sent under the session.

Table 28 Get channel cipher suites command request and response data

IPMI Request Data Byte	Data field	
1	Channel Number	
	[7:4]	Reserved
	[3:0]	Channel number 0h-Bh, Fh = Channel numbers Eh = retrieve information for channel this request was issued on.
2	Payload Type. [7:6] - reserved [5:0] - Payload Type number Typically 00h (IPMI). The Payload Type number is used to look up the Security Algorithm support when establishing a separate session for a given payload type.	
3	List Index.	
	[7]	1b = list algorithms by Cipher Suite 0b = list supported algorithms ¹
	[6]	Reserved
	[5:0]	List index (00h-3Fh). 0h selects the first set of 16, 1h selects the next set of 16, and so on. 00h = Get first set of algorithm numbers. The MC returns sixteen (16) bytes at a time per index, starting from index 00h, until the list data is exhausted, at which point it will 0 bytes or <16 bytes of list data.
IPMI Response Data Byte	Data field	
1	Completion Code	
2	Channel Number Channel number that the Authentication Algorithms are being returned for. If the channel number in the request was set to Eh, this will return the channel number for the channel that the request was received on.	
(3:18)	Cipher Suite Record data bytes, per Table 22-18, Cipher Suite Record	

Table 28 Get channel cipher suites command request and response data *(continued)*

	<p>Format. Record data is 'packed'; there are no pad bytes between records. It is possible that record data will span across multiple List Index values.</p> <p>The MC returns sixteen (16) bytes at a time per index, starting from index 00h, until the list data is exhausted, at which point it will 0 bytes or <16 bytes of list data.</p>
--	--

¹ When listing numbers for supported algorithms, the MC returns a list of the algorithm numbers for each algorithm that the MC supports on a given channel. Each algorithm is listed consecutively and only listed once. There is no requirement that the MC return the algorithm numbers in any specific order.

Cipher suite records

The data from the Get Channel Cipher Suites command is issued as Cipher Suite records. Tag bits are used to delimit different fields in the record. Each record starts off with a "Start Of Record" byte. This byte can be 30h or 31h, indicating that the Start Of Record byte is followed either by an Cipher Suite ID, or by a OEM Cipher Suite ID plus OEM IANA.

Following the header bytes are algorithm number bytes for the different algorithms that form the Cipher Suite. Each byte is tagged with the type of algorithm the number is for. Cipher Suite records are required to list algorithms in the order: Authentication Algorithm number first, Integrity Algorithm numbers next, and Confidentiality Algorithm numbers last.

If more than one algorithm of a given type is listed in the Cipher Suite Record, then any one of the algorithms can be used in combination with the other types. For example, if a Cipher Suite response returns both MD5 and MD2 as Authentication and Integrity algorithms, and xRC4 for confidentiality, then the allowed combinations are [MD2, MD2, xRC4], [MD2, MD5, xRC4], [MD5, MD2, xRC4], and [MD5, MD5, xRC4]. A remote console can negotiate for those combinations when establishing a session.

Table 29 Cipher suite record format

Size	Tag bits [7:6]	Tag bits [5:0]
2 or 5		<p>This field starts off with either a C0h or C1h "Start of Record" byte, depending on whether the Cipher Suite is a standard Cipher Suite ID or an OEM Cipher Suite, respectively.</p> <p>Standard cipher suite ID</p> <ul style="list-style-type: none"> Byte 1: [7:0] = 1100_0000b. Start of Record, Standard Cipher Suite. Byte 2 (Data following C0h (1100_0000b) start of record byte): Cipher Suite ID—This value is used a numeric way of identifying the Cipher Suite on the platform. It's used in commands and configuration parameters that enable and disable Cipher Suites. <p>OEM cipher suite</p> <ul style="list-style-type: none"> Byte 1: [5:0] = 1100_0001b — Start of Record, OEM Cipher Suite. Byte 2 (Data following C1h (1100_0001) start of record byte): OEM Cipher Suite ID <p>Byte 3:5 - OEM IANA Least significant byte first. 3-byte IANA for the OEM or body that defined the Cipher Suite.</p>
1	00b	<p>[5:0] = Authentication Algorithm Number. A Cipher Suite is only allowed to utilize one Authentication algorithm.</p>

Table 29 Cipher suite record format *(continued)*

Size	Tag bits [7:6]	Tag bits [5:0]
var	01b	[5:0] = Integrity Algorithm Number(s).
var	10b	[5:0] = Confidentiality Algorithm Number(s).

Cipher suite ID numbers

The following table provides the number ranges and assignments for Cipher Suite IDs. The Cipher Suite ID values are used as a way to identify different Cipher Suites in configuration parameters and IPMI commands.

The OEM IDs do not correspond to a particular Cipher Suite, but are handles that can be used to identify the Cipher Suite on a particular implementation of a MC. I.e. the OEM Cipher Suite corresponding to “80h” can be different from one MC to the next. These handles can, however, be used in configuration parameters and commands the same way as the IPMI-defined Cipher Suite IDs.

The Get Channel Cipher Suites command will return the algorithms used to form a given Cipher Suite (those numbers can then be used by a remote console in the commands for establishing a session). For OEM defined Cipher Suites, the Get Channel Cipher Suites command will also return the IANA for the OEM or body that defined the Cipher Suite.

Table 30 Cipher suite ID numbers

ID	Characteristics	Cipher Suite	Authentication Algorithm	Integrity Algorithm(s)	Confidentiality Algorithm(s)
0	no password	00h, 00h, 00h	RAKP-none	None	None
1	S	01h, 00h, 00h	RAKP-HMAC-SHA1	None	None
2	S, A	01h, 01h, 00h		HMAC-SHA1-96	None
3	S, A, E	01h, 01h, 01h			AES-CBC-128
4	S, A, E	01h, 01h, 02h			xRC4-128
5	S, A, E	01h, 01h, 03h			xRC4-40
6	S	02h, 00h, 00h	RAKP-HMAC-MD5	None	None
7	S, A	02h, 02h, 00h		HMAC-MD5-128	None
8	S, A, E	02h, 02h, 01h			AES-CBC-128
9	S, A, E	02h, 02h, 02h			xRC4-128
10	S, A, E	02h, 02h, 03h			xRC4-40
11	S, A	02h, 03h, 00h		MD5-128	None
12	S, A, E	02h, 03h, 01h			AES-CBC-128
13	S, A, E	02h, 03h, 02h			xRC4-128
14	S, A, E	02h, 03h, 03h			xRC4-40
80h-BFh	OEM specified	OEM specified	OEM specified	OEM specified	OEM specified
C0h-FFh	reserved	-	-	-	-
Key: S = Authenticated session setup (correct role, username and password/key required to establish session.)					

Table 30 Cipher suite ID numbers *(continued)*

ID	Characteristics	Cipher Suite	Authentication Algorithm	Integrity Algorithm(s)	Confidentiality Algorithm(s)
	A = Authenticated payload data supported. E = Authentication and encrypted payload data supported.				

Set session privilege level command

This command is available to the MC.

This command is sent in authenticated format. When a session is activated, the session is set to an initial privilege level. A session that is activated at a maximum privilege level of callback is set to an initial privilege level of callback and cannot be changed. All other sessions are initially set to user level, regardless of the maximum privilege level requested in the `activate session` command. The remote console must raise the privilege level of the session using this command in order to execute commands that require a greater-than-user level of privilege.

This command cannot be used to set a privilege level higher than the lowest of the privilege level set for the user (via the `set user access` command) and the privilege limit for the channel that was set via the `set channel access` command. The specification allows a session to be used across multiple channels. The maximum privilege limit and authentication are based on the user privilege and channel limits. Since these can vary on a per channel basis, an implementation cannot simply assign a single privilege limit to a given session but must authenticate incoming messages according to the specific settings for the channel and the user on a per-channel basis.

Table 31 Set session privilege level command request and response data

IPMI request data byte number	Data field
1	Requested privilege level <ul style="list-style-type: none"> • [7:4] — Reserved • [3:0] — Privilege level <ul style="list-style-type: none"> ◦ 0h — No change, just return present privilege level ◦ 1h — Reserved ◦ 2h — Change to user level ◦ 3h — Change to operator level ◦ 4h — Change to administrator level ◦ 5h — Change to OEM proprietary level ◦ All other = Reserved
IPMI response data byte number	Data field
1	Completion code. Generic, plus following command specific: <ul style="list-style-type: none"> • 80h = Requested level not available for this user • 81h = Requested level exceeds channel and/or user privilege limit • 82h = Cannot disable user level authentication
2	New privilege level (or present level if return present privilege level was selected.)

Close session command

This command is used to immediately terminate a session in progress. It is typically used to close the session that the user is communicating over, though it can be used to terminate other sessions in progress (provided that the user is operating at the appropriate privilege level, or the command is executed over a local channel, such as the system interface).

Table 32 Close session command request and response data

IPMI request data byte number	Data field
1:4	Session ID. For IPMI v2.0/RMCP+ this is the Managed System Session ID value that was generated by the MC, not the ID from the remote console. If Session ID = 0000_0000h then an implementation can optionally enable this command to take an additional byte of parameter data that allows a session handle to be used to close a session.
(5)	Session Handle. Only present if Session ID = 0000_0000h.
IPMI response data byte number	Data field
1	Completion code. <ul style="list-style-type: none">• 87h = Invalid session ID in request• 88h = Invalid Session Handle in request• 82h = Cannot disable user level authentication

Get session info command

This command is available to the MC.

This command is used to get information regarding which users presently have active sessions, and, when available, addressing information for the party that has established the session. A portion of the response is dependent on the type of channel.

For IPMI v2.0, a previously reserved field has been defined to hold a value indicating whether a session operating on a channel of channel type = 802.3 LAN is presently using IPMI v1.5 or v2.0/RMCP+ protocols.

Table 33 Get session info command request and response data

IPMI request data byte number	Data field
1	Session index. This value is used to select entries in a logical sessions table maintained by the management controller. Information for all active sessions can be retrieved by incrementing the session index from 1 to N, where N is the number of entries in the active sessions table. <ul style="list-style-type: none">• 00h = Return information for the active session associated with the session where this command was received.• N = Get information for Nth active session• FEh = Look up session information according to the session handle passed in this request.• FFh = Look up session information according to the session ID passed in this request.
Present if session index = FEh	
2	Session handle. 00h = reserved.
Present if session index = FFh:	
2:5	Session ID. ID of session to look up session information. For IPMI v2.0/RMCP+ this is the session ID value that was generated by the MC, not the ID from the remote console.

Table 33 Get session info command request and response data *(continued)*

IPMI response data byte number	Data field
1	Completion code
2	Session handle presently assigned to active session. FFh = reserved. Return 00h if no active session associated with given session index.
3	Number of possible active sessions. This value reflects the number of possible entries (slots) in the sessions table. <ul style="list-style-type: none"> • [7:6] — Reserved • [5:0] — Session slot count. 1-based.
4	Number of currently active sessions on all channels on this controller. <ul style="list-style-type: none"> • [7:6] — Reserved • [5:0] — Active session count. 1-based. 0=no currently active sessions.
The following parameters are returned only if there is an active session corresponding to the given session index:	
5	User ID for selected active session. <ul style="list-style-type: none"> • [7:6] — Reserved • [5:0] — User ID. 000000b = reserved.
6	Operating privilege level <ul style="list-style-type: none"> • [7:4] — Reserved • [3:0] — Present privilege level at which the user is operating.
7	[7:4] — Session protocol auxiliary data. For channel type = 802.3 LAN: <ul style="list-style-type: none"> • 0h = IPMI v1.5 • 1h = IPMI v2.0/RMCP+ Channel in which the session was activated. [3:0] - Channel number
The following bytes 8:18 are optionally returned if channel type = 802.3 LAN:	
8:11	IP address of remote console (MS-byte first). Address that was received in the <code>activate session</code> command that activated the session
12:17	MAC address (MS-byte first). Address that was received in the <code>activate session</code> command that activated the session.
18:19	Port number of remote console (LS-byte first). Port number that was received in UDP packet that held the <code>activate session</code> command that activated the session (for IPMI v1.5 packets) or that was used for in the packet for RAKP Message 3 (for IPMI v2.0 / RMCP+ packets).
The following bytes 8:13 are returned if channel type = asynch. serial/modem:	
8	Session/channel activity type: <ul style="list-style-type: none"> • 0 = IPMI messaging session active • 1 = Callback messaging session active • 2 = Dial-out alert active • 3 = TAP page active
9	Destination selector for active call-out session. 0 otherwise. <ul style="list-style-type: none"> • [7:4] - Reserved • [3:0] - Destination selector. Destination 0 is always present as a volatile destination that is used with the <code>alert immediate</code> command.

Table 33 Get session info command request and response data *(continued)*

10:13	If PPP connection, the IP address of the remote console. (MS-byte first). 00h, 00h, 00h, 00h otherwise.
The following additional bytes 14:15 are returned if channel type = asynch. serial/modem and connection is PPP:	
14:15	Port address of remote console (LS-byte first). Address that was received in the <code>activate session</code> command that activated the session.

Get AuthCode command

This command is available to the MC.

This command is used to send a block of data to the MC, whereupon the MC returns a hash of the data together concatenated with the internally stored password for the given channel and user. This command allows a remote console to send an AuthCode and data block to system software on a remote platform, whereby the system software can validate the AuthCode by comparing it with the AuthCode returned by the MC. This enables the MC to serve as a validation agent for remote requests that come through local system software instead of through a remote session directly with the MC.

The following is an outline of potential use of this capability. Remote console software could request that system software perform a particular operation. In response, local system software could deliver a challenge string to the remote console, which would be required to hash it with the desired password and return the AuthCode to the local system software. The local system software would then perform the requested operation only if it found that the AuthCode matched the one returned by the MC. The local software would typically implement mechanisms to bind the challenge string to the requested operation to ensure that the challenge string and AuthCode combination only applied to a given instance of the requested operation, and even from a particular remote console.

- Managed system delivers a random number token, *S*, to the console. In this example, the console uses *S* to identify a particular request. The managed system tracks outstanding *S* values, and expires them either because a valid message was received from a console that used that token, or because the token was not used within a specified interval.
- Console determines: *X* = data to be authenticated
 - $K1 = 16\text{-byte signature of } X \text{ and a sequence number} = \text{hash}(X, S, \text{SW_Authentication_Type})$. Where *SW_Authentication_Type* is any signature algorithm management software wishes to use for providing a signature given *X* and *S*.
 - $K2 = 16\text{-byte hash of } K1 \text{ and the password} = \text{hash}(K1, \text{PWD}, \text{Authentication_Type})$. Where *Authentication_Type* in this case is one of the supported authentication types for the given MC.
- Console sends *X*, *S*, and *K2* to software agents on the managed system.
- Software agent on the managed system calculates *K1* from *X* and *S* that it received by locally calculating $K1 = \text{hash1}(X, S, \text{SW_Authentication_Type})$. The software also verifies that *S* is a valid outstanding token.
- Managed system passes *K1* to MC. MC internally looks up password based on the user ID passed in the `get authcode` command and produces: $K2_{\text{BMC}} = \text{hash}(K1, \text{PWD}, \text{Authentication_Type})$.
- Managed system accepts data if software agents finds that $K2 = K2_{\text{BMC}}$.

Table 34 Get AuthCode command request and response data

IPMI request data byte number	Data field
1	<p>[7:6] - Authentication type / Integrity algorithm number</p> <ul style="list-style-type: none"> • 00b = IPMI v1.5 AuthCode algorithms • 01b = IPMI v2.0/RMCP+ algorithm number <p>For [7:6] = 00b, IPMI v1.5 AuthCode number:</p> <ul style="list-style-type: none"> • [5:4] - Reserved • [3:0] - Hash type <ul style="list-style-type: none"> ◦ 0h = Reserved ◦ 1h = MD2 ◦ 2h = MD5 ◦ 3h = Reserved ◦ 4h = Reserved (change from IPMI v1.5). This results in an error completion code. ◦ 5h = OEM proprietary ◦ All other = Reserved <p>For [7:6] = 01b, IPMI v2.0/RMCP+ Integrity algorithm number</p> <ul style="list-style-type: none"> • 5:0] - Integrity algorithm number. The user password is used as the starting key for the Integrity algorithm, instead of session-dependent keys such as the session integrity key. The "none" Integrity number (0) is illegal and results in an error completion code.
2	<p>Channel number</p> <ul style="list-style-type: none"> • [7:4] - Reserved • [3:0] - Channel number
3	User ID. (Software will typically have to use the <code>get user name</code> command to look up the user ID from a user name).
4:19	Data to hash (must be 16 bytes).
IPMI response data byte number	Data field
1	Completion code.
For IPMI v1.5 AuthCode number:	
2:17	AuthCode =
For IPMI v2.0 Integrity algorithm number	
(2:21)	Resultant hash, per selected Integrity algorithm. Up to 20 bytes. An implementation can elect to return a variable length field based on the size of the hash for the given integrity algorithm, or can return a fixed field where the hash data is followed by 00h bytes as needed to pad the data to 20 bytes.

Set channel access command

This command is available to the MC.

This command is used to configure whether channels are enabled or disabled, whether alerting is enabled or disabled for a channel, and to set which system modes channels are available under. This configuration is saved in non-volatile storage associated with the MC. The choice of factory default setting for the non-volatile parameters is left to the implementer or system integrator.

The active (volatile) settings can be overwritten to allow run-time software to make temporary changes to the access. The volatile settings are overwritten from the non-volatile settings whenever the system is reset or transitions to a powered off state.

An implementation can elect to provide a subset of the possible access mode options. If a given access mode is not supported, the command-specific completion code 83h (access mode not supported) must be returned.

Table 35 Set channel access command request and response data

Request data byte number	Data field		
1	[7:4] — Reserved [3:0] — Channel number		
2	[7:6]	00b =	Do not set or change channel access.
		01b =	Set non-volatile channel access according to bits [5:0].
		10b =	Set volatile (active) setting of channel access according to bits [5:0].
		11b =	Reserved.
	[5]	PEF alerting enable/disable. This bit globally gates whether PEF alerts can be issued from the given channel. Setting this to enable PEF alerting is a necessary part of enabling alerts for the channel, but for alerts to be generated, the PEF and channel configuration must also be set to enable alerting. Setting this bit to enable does not alter the PEF configuration or the alerting settings in the channel's configuration parameters. For example, if PEF is not configured for generating an alert, enabling PEF alerting with this bit does not change that configuration. Setting this bit to disable blocks PEF-generated alerts regardless of the PEF and channel configuration parameters.	
		0b =	Enable PEF alerting.
		1b =	Disable PEF alerting on this channel. The <code>alert immediate</code> command can still be used to generate alerts.
	[4]	Per-message authentication enable/disable. This bit is ignored for channels (such as serial/modem) that do not support per-message authentication.	
		0b =	Enable per-message authentication.
		1b =	Disable per-message authentication. Authentication is required to activate any session on this channel, but authentication is not used on subsequent packets for the session.
	[3]	User level authentication enable/disable. Optional. Return a CCh invalid data field error completion code if an attempt is made to set this bit, but the option is not supported.	
		0b =	Enable user level authentication. All user level commands are to be authenticated per the authentication type that was negotiated when the session was activated.
		1b =	Disable user level authentication. Allow user level commands to be executed without being authenticated. If the option to disable user level command authentication is accepted, the MC will accept packets with authentication type set to none if they contain user level commands. For outgoing packets, the MC returns responses with the same authentication type that was used for the request.
	[2:0]	Access mode for IPMI messaging. (PEF alerting is enabled/disabled separately from IPMI messaging, see bit 5).	
		000b =	Disabled. Channel disabled for IPMI messaging.
		001b =	Pre-boot only. Channel only available when system is in a powered down state or in BIOS before start of boot.

Table 35 Set channel access command request and response data *(continued)*

		010b =	Always available. Channel always available for communication regardless of system mode. BIOS typically dedicates the serial connection to the MC.
		011b =	Shared. Same as always available, but BIOS typically leaves the serial port available for software use.
3	Channel privilege level limit. This value sets the maximum privilege level that can be accepted on the specified channel.		
	[7:6]	00b =	Do not set or change channel privilege level limit.
		01b =	Set non-volatile privilege level limit according to bits [3:0].
		10b =	Set volatile setting of privilege level limit according to bits [3:0].
		11b =	Reserved.
	[5:4]	Reserved.	
	[3:0]	Channel privilege level limit.	
		0h =	Reserved.
		1h =	Callback level.
		2h =	User level.
		3h =	Operator level.
		4h =	Administrator level.
		5h =	OEM proprietary level.
Response data byte number	Data field		
1	Completion code. Generic, plus the following command-specific completion codes:		
	82h =	Set not supported on selected channel (for example, channel is session-less).	
	83h =	Access mode not supported.	

Get channel access command

This command is available to the MC.

This command is used to return whether a given channel is enabled or disabled, whether alerting is enabled or disabled for the entire channel, and under what system modes the channel can be accessed.

Table 36 Get channel access command request and response data

Request data byte number	Data field		
1	[7:4] — Reserved [3:0] — Channel number		
2	[7:6]	00b =	Reserved
		01b =	Get non-volatile channel access
		10b =	Get present volatile (active) setting of channel access
		11b =	Reserved
	[5:0]	Reserved	
Response data byte number	Data field		

Table 36 Get channel access command request and response data *(continued)*

1	Completion code. Generic, plus the command-specific completion code:	
	82h =	Command not supported for selected channel (for example, the channel is session-less.)
2	[7:6]	Reserved.
	[5]	0b = Alerting enabled.
		1b = Alerting disabled.
	[4]	Per-message authentication enable/disable. This bit is unspecified for channels (such as serial/modem) that do not support per-message authentication.
		0b = Per message authentication enabled.
		1b = Per message authentication disabled.
	[3]	User level authentication enable
		0b = User level authentication enabled.
		1b = User level authentication disabled.
	[2:0]	Access mode
		0h = Disabled. Channel disabled for communication.
		1h = Pre-boot only. Channel only available when system is in a powered down state or in BIOS before start of boot.
		2h = Always available. Channel always available for communication regardless of system mode. BIOS typically dedicates the serial connection to the MC.
		3h = Shared. Same as always available, but BIOS typically leaves the serial port available for software use.
3	Channel privilege level limit. This value returns the maximum privilege level that can be accepted on the specified channel.	
	[7:4]	Reserved.
	[3:0]	Channel privilege level limit.
		0h = Reserved.
		1h = Callback level.
		2h = User level.
		3h = Operator level.
		4h = Administrator level.
		5h = OEM proprietary level.

Get channel info command

This command returns media and protocol information about the given channel. The channel protocol may vary with changes to the configuration parameters associated with the channel.

Table 37 Get channel info command request and response data

IPMI request data byte number	Data field
1	[7:4] — Reserved [3:0] — Channel number. Use Eh to get information about the channel from which this command is being executed.

Table 37 Get channel info command request and response data *(continued)*

IPMI response data byte number	Data field		
1	Completion code		
2	[7:4]	Reserved	
	[3:0]	Actual channel number. This value typically matches the channel number passed in the request, unless the request is for channel E, in which case the response returns the actual channel number.	
3	[7:4]	Reserved	
	[6:0]	7-bit channel medium type	
4	Channel protocol type:		
	[7:5]	Reserved	
	[4:0]	5-bit channel IPMI messaging protocol type	
5	Session support		
	[7:6]	00b =	Channel is session-less
		01b =	Channel is single-session
		10b =	Channel is multi-session
		11b =	Channel is session-based (return this value if a channel could alternate between single- and multi-session operation, as can occur with a serial/modem channel that supports connection mode auto-detect)
	Number of sessions that have been activated on the given channel.		
[5:0]	Active session count. 1-based. 00_0000b = no sessions have been activated on this channel.		
6	Vendor ID (IANA enterprise number) for OEM/organization that specified the channel protocol. Least significant byte first. Returns the IPMI IANA for IPMI-specification defined, non-OEM protocol type numbers other than OEM. The IPMI enterprise number is: 7154 (decimal). This gives the values F2h, 1Bh, 00h for bytes 6 through 8, respectively. This value is returned for all channel protocols specified in this document, including PPP.		
9:10	Auxiliary channel info For channel = Fh (system interface): <ul style="list-style-type: none">Byte 1: SMS interrupt type<ul style="list-style-type: none">00h-0Fh = IRQ 0 through 15, respectively10h-13h = PCI A-D, respectively14h = SMI15h = SCI20h-5Fh = System interrupt 0 through 63, respectively60h = Assigned by ACPI / Plug 'n Play BIOSFFh = No interrupt / unspecifiedAll other = ReservedByte 2: Event message buffer interrupt type. See values for byte 1.		

Table 37 Get channel info command request and response data *(continued)*

	For OEM channel types: Byte 1:2 = OEM specified per OEM identified by vendor ID. All other channel types: Byte 1:2 = reserved.
--	---

Set user access command

This command is available to the MC.

This command is used to configure the privilege level and channel accessibility associated with a given user ID. If this command is not supported, then a single null user (User 1) per channel is assumed and the privilege level and channel access are determined solely by the settings returned by the `get channel access limits` command. If implemented, this command must support at least the null user (User 1). The number of additional users supported is left to the implementer.

NOTE: The limits set using the `set channel access` command take precedence over the `set user access` command settings. That is, if a given channel is limited to user level then all users are limited to user level operation regardless of what their user access levels were set to using the `set user access` command. Changes made to the user access and privilege levels may not take affect until the next time the user establishes a session.

Table 38 Set user access command request and response data

Request data byte number	Data field		
1	[7]	0b =	Do not change any of the following bits in this byte.
		1b =	Enable changing the following bits in this byte.
	[6]	User restricted to callback	
		0b =	User privilege limit is determined by the user privilege limit parameter, below, for both callback and non-callback connections.
		1b =	User privilege limit is determined by the user privilege limit parameter for callback connections, but is restricted to callback level for non-callback connections. Thus, a user can only initiate a callback when they call in to the MC, but once the callback connection has been made, the user could potentially establish a session as an operator.
	[5]	User link authentication enable/disable (used to enable whether this user's name and password information will be used for link authentication, for example PPP CHAP) for the given channel. Link authentication itself is a global setting for the channel and is enabled/disabled via the serial/modem configuration parameters.	
		<ul style="list-style-type: none"> 0b = disable user for link authentication 1b = enable user for link authentication 	
	[4]	User IPMI messaging enable/disable (used to enable/disable whether this user's name and password information will be used for IPMI messaging. In this case, IPMI messaging refers to the ability to execute generic IPMI commands that are not associated with a particular payload type. For example, if IPMI messaging is disabled for a user, but that user is enabled for activating the SOL payload type, then IPMI commands associated with SOL and session management, such as <code>get SOL configuration parameters</code> and <code>close session</code> are available, but generic IPMI commands such as <code>get SEL time</code> are unavailable.)	
		<ul style="list-style-type: none"> 0b = Disable user for IPMI messaging 1b = Enable user for IPMI messaging 	
	[3:0]	Channel number	

Table 38 Set user access command request and response data *(continued)*

2	User ID <ul style="list-style-type: none"> [7:6] — Reserved [5:0] — User ID. 000000b = Reserved. 	
3	User limits	
	[7:4]	Reserved
	[3:0]	User privilege limit. Determines the maximum privilege level that the user is allowed to switch to on the specified channel.
		0h = Reserved
		1h = Callback
		2h = User
		3h = Operator
		4h = Administrator
		5h = OEM proprietary
		Fh = No access
(4)	User session limit (optional). Sets how many simultaneous sessions can be activated with the username associated with this user. If not supported, the username can be used to activate as many simultaneous sessions as the implementation supports. Return a CCh invalid data field error completion code if an attempt is made to set a non-zero value in this field, but the option is not supported.	
	[7:4]	Reserved
	[3:0]	User simultaneous session limit. 1-based. 0h = only limited by the implementations overall support for simultaneous sessions.
Response data byte number	Data field	
1	Completion code. NOTE: An implementation does not return an error completion code if the user access level is set higher than the privilege limit for a given channel. To bring attention to this condition, the software must check the channel privilege limits set using the <code>set channel access</code> command and provide notification of any mismatch.	

Get user access command

This command is available to the MC.

This command is used to retrieve channel access information and enabled/disabled state for the given user ID. The command also returns information about the number of supported users.

Table 39 Get user access command request and response data

Request data byte number	Data field	
1	[7:4]	Reserved
	[3:0]	Channel number
2	[7:6]	Reserved
	[3:0]	User ID. 000000b = reserved.
Response data byte number	Data field	
1	Completion code.	

Table 39 Get user access command request and response data *(continued)*

	NOTE: An implementation does not return an error completion code if the user access level is set higher than the privilege limit for a given channel. To bring attention to this condition, the software must check the channel privilege limits and provide notification of the mismatch.																																											
2	Maximum number of user IDs. 1-based. Count includes User 1. A value of 1 indicates only User 1 is supported. <ul style="list-style-type: none">• [7:6] — Reserved• [5:0] — Maximum number of user IDs on this channel.																																											
3	Count of currently enabled user IDs (1-based). A value of 0 indicates that all users, including User 1, are disabled. This is equivalent to disabling access to the channel. <table><tr><td>[7:6]</td><td colspan="2">User ID enable status (for IPMI v2.0 errata 3 and later implementations).</td></tr><tr><td></td><td>00b =</td><td>User ID enable status unspecified. (For backward compatibility with pre-errata 3 implementations. IPMI errata 3 and later implementations should return the 01b and 10b responses.)</td></tr><tr><td></td><td>01b =</td><td>User ID enabled via <code>set user password</code> command.</td></tr><tr><td></td><td>10b =</td><td>User ID disabled via <code>set user password</code> command.</td></tr><tr><td></td><td>11b =</td><td>Reserved.</td></tr><tr><td>[5:0]</td><td colspan="2">Count of currently enabled user IDs on this channel which indicates how many user ID slots are presently in use.</td></tr></table>		[7:6]	User ID enable status (for IPMI v2.0 errata 3 and later implementations).			00b =	User ID enable status unspecified. (For backward compatibility with pre-errata 3 implementations. IPMI errata 3 and later implementations should return the 01b and 10b responses.)		01b =	User ID enabled via <code>set user password</code> command.		10b =	User ID disabled via <code>set user password</code> command.		11b =	Reserved.	[5:0]	Count of currently enabled user IDs on this channel which indicates how many user ID slots are presently in use.																									
[7:6]	User ID enable status (for IPMI v2.0 errata 3 and later implementations).																																											
	00b =	User ID enable status unspecified. (For backward compatibility with pre-errata 3 implementations. IPMI errata 3 and later implementations should return the 01b and 10b responses.)																																										
	01b =	User ID enabled via <code>set user password</code> command.																																										
	10b =	User ID disabled via <code>set user password</code> command.																																										
	11b =	Reserved.																																										
[5:0]	Count of currently enabled user IDs on this channel which indicates how many user ID slots are presently in use.																																											
4	Count of user IDs with fixed names, including User 1 (1-based). Fixed names in addition to User 1 are required to be associated with sequential user IDs starting from User ID 2. <ul style="list-style-type: none">• [7:6] — Reserved• [5:0] — Count of user IDs with fixed names on this channel.																																											
5	Channel access <table><tr><td>[7]</td><td colspan="2">Reserved</td></tr><tr><td>[6]</td><td>0b =</td><td>User access available during call-in or callback direct connection.</td></tr><tr><td></td><td>1b =</td><td>User access available only during callback connection.</td></tr><tr><td colspan="3">For pre- IPMI v2.0 errata 3 implementations: bits 5:4 are used for determining the count of currently enabled user IDs in byte 3. Either bit being set to 1b represents an enabled user ID. For IPMI v2.0 errata 3 and later implementations: the count of enabled User IDs is based on the user IDs that are presently enabled as reflected in byte 3, bits [7:6], user ID enable status.</td></tr><tr><td colspan="3">NOTE: Some pre- IPMI v2.0 errata 3 implementations may automatically clear bits [5:4], and may also prevent them from being set, while the user ID is disabled. IPMI v2.0 errata 3 and later implementations should not alter bits [5:4] based on whether or not a user ID is enabled.</td></tr><tr><td>[5]</td><td>0b =</td><td>User disabled for link authentication</td></tr><tr><td></td><td>1b =</td><td>User enabled for link authentication</td></tr><tr><td>[4]</td><td>0b =</td><td>User disabled for IPMI messaging</td></tr><tr><td></td><td>1b =</td><td>User enabled for IPMI messaging</td></tr><tr><td>[3:0]</td><td colspan="2">User privilege limit for given channel</td></tr><tr><td></td><td>0h =</td><td>Reserved</td></tr><tr><td></td><td>1h =</td><td>Callback</td></tr><tr><td></td><td>2h =</td><td>User</td></tr><tr><td></td><td>3h =</td><td>Operator</td></tr></table>		[7]	Reserved		[6]	0b =	User access available during call-in or callback direct connection.		1b =	User access available only during callback connection.	For pre- IPMI v2.0 errata 3 implementations: bits 5:4 are used for determining the count of currently enabled user IDs in byte 3. Either bit being set to 1b represents an enabled user ID. For IPMI v2.0 errata 3 and later implementations: the count of enabled User IDs is based on the user IDs that are presently enabled as reflected in byte 3, bits [7:6], user ID enable status.			NOTE: Some pre- IPMI v2.0 errata 3 implementations may automatically clear bits [5:4], and may also prevent them from being set, while the user ID is disabled. IPMI v2.0 errata 3 and later implementations should not alter bits [5:4] based on whether or not a user ID is enabled.			[5]	0b =	User disabled for link authentication		1b =	User enabled for link authentication	[4]	0b =	User disabled for IPMI messaging		1b =	User enabled for IPMI messaging	[3:0]	User privilege limit for given channel			0h =	Reserved		1h =	Callback		2h =	User		3h =	Operator
[7]	Reserved																																											
[6]	0b =	User access available during call-in or callback direct connection.																																										
	1b =	User access available only during callback connection.																																										
For pre- IPMI v2.0 errata 3 implementations: bits 5:4 are used for determining the count of currently enabled user IDs in byte 3. Either bit being set to 1b represents an enabled user ID. For IPMI v2.0 errata 3 and later implementations: the count of enabled User IDs is based on the user IDs that are presently enabled as reflected in byte 3, bits [7:6], user ID enable status.																																												
NOTE: Some pre- IPMI v2.0 errata 3 implementations may automatically clear bits [5:4], and may also prevent them from being set, while the user ID is disabled. IPMI v2.0 errata 3 and later implementations should not alter bits [5:4] based on whether or not a user ID is enabled.																																												
[5]	0b =	User disabled for link authentication																																										
	1b =	User enabled for link authentication																																										
[4]	0b =	User disabled for IPMI messaging																																										
	1b =	User enabled for IPMI messaging																																										
[3:0]	User privilege limit for given channel																																											
	0h =	Reserved																																										
	1h =	Callback																																										
	2h =	User																																										
	3h =	Operator																																										

Table 39 Get user access command request and response data *(continued)*

	4h =	Administrator
	5h =	OEM proprietary
	Fh =	No access. This value does not add to, or subtract from, the number of enabled user IDs

Set user name command

This command is available to the MC.

This command adds a new user ID. The names are stored as a logical array within non-volatile storage associated with the management controller. Names are stored and retrieved using the user ID as the index into the logical array. There is no configurable name for User ID 1. User ID 1 is reserved for the null user name, User 1. Null user is not supported.

The management controller does not prevent duplicate user names from being enabled for the same channel. It is the responsibility of configuration software to ensure that duplicate user names are not enabled simultaneously for the same channel.

Having duplicate user names does not cause functional problems with the MC because the MC uses the first username match that it finds. However, it could be confusing to the user if they have duplicate user names enabled for a given channel, since only the settings for the first encountered user name would be used by the MC.

This command is highly recommended for session-based channels. It is also recommended that the implementation support multiple users with configurable user names.

Table 40 Set user name command request and response data

Request data byte number	Data field
1	User ID <ul style="list-style-type: none"> [7:6] — Reserved [5:0] — User ID. 000000b = reserved. (User ID 1 is permanently associated with User 1, the null user name).
2:17	User name string in ASCII, 16 bytes, max. Strings with fewer than 16 characters are terminated with a null (00h) character and 00h padded to 16 bytes. When the string is read back using the <code>get user name</code> command, those bytes return as 0's.
Response data byte number	Data field
1	Completion code.

Get user name command

This command is available to the MC.

This command is used to retrieve user name information that was set using the `set user name` command. Configuration software can use this command to retrieve user names.

Table 41 Get user name command request and response data

Request data byte number	Data field
1	User ID <ul style="list-style-type: none"> [7:6] — Reserved [5:0] — User ID. 000000b = reserved.

Table 41 Get user name command request and response data *(continued)*

Response data byte number	Data field
1	Completion code.
2:17	User name string in ASCII, 16 bytes, max. Strings with fewer than 16 characters are terminated with null (00h) characters filling in the remaining bytes. MC does not check to see whether string data is printable or not. Only character that MC interprets is null (00h).

Set user password command

This command is available to the MC.

This command is used to set and change user passwords and to enable and disable user IDs. If no password protection is desired for a given user, the password must be stored as an ASCII null-string. The management controller firmware forces the remaining fifteen bytes to 00h and stores the password as sixteen bytes of 00h.

The password is stored as a 16-byte or 20-byte (for IPMI v2.0/RMCP+) octet string. All values (0-255) are allowed for every byte. The management controller does not check the format or interpret values that are passed in with this command.

Software is allowed to place additional restrictions on what passwords can be entered, in which case it is the responsibility of configuration software and console software to stay in synch with that definition. For example, remote console software could restrict passwords to the printable ASCII character set in order to simplify direct keyboard entry. If this is done, any companion configuration utility should ensure that the user does not configure the managed system with non-printable passwords. Otherwise, it would be possible for the management controller to be configured with passwords that could not be entered via the remote console utility.

Table 42 Set user password command request and response data

Request data byte number	Data field
1	User ID For IPMI v2.0, the MC supports 20-byte passwords for all supported user IDs that have configurable passwords. The MC maintains an internal tag that indicates whether the password was set as a 16-byte or as a 20-byte password. A 16-byte password can be used in algorithms that call for a 20-byte password. In this case, the 16-byte password is padded with 0's to 20- bytes. The test password operation returns the test failed error completion code if an attempt is made to test a password that was stored as a 20-byte password as a 16-byte password (per password size bit 7), and vice versa. The test password operation can be used to determine whether a password has been stored as 16-bytes or 20-bytes. A password that has been stored as a 20-byte password cannot be used for establishing an IPMI v1.5 session. If it is necessary to configure the same password for both IPMI v2.0 and IPMI v1.5 access, it must be set as a 16-byte password. ¹ The password is padded with 0's as necessary for IPMI v2.0 / RMCP+ use.
	[7]
	Password size
	1b =
	Set 20-byte user password/key.
	0b =
	Set 16-byte user password/key (IPMI v1.5 backward compatible).
2	[6]
	Reserved
	[5:0]
	User ID. 000000b = reserved. (User ID 1 is permanently associated with User 1, the null user name).
2	[7:2]
	Reserved
	[1:0]
	Operation

Table 42 Set user password command request and response data *(continued)*

		00b =	Disable user
		01b =	Enable user
		10b =	Set password
		11b =	Test password. Compares the password data given in the request with the presently stored password and returns an OK completion code if there is a match. Otherwise, an error completion code is returned. (See the completion code description in the response data.)
For password size = 16 bytes:			
3:18	Password data. This is a required, fixed length field when used for the set and test password operations. If the password is entered as an ASCII string, it must be null (00h) terminated and 00h padded if the string is shorter than 16 bytes. This field is not needed if the operation is disable user or enable user. If this field is present for those operations, the MC ignores the data.		
For password size = 20 bytes:			
3:22	Password data. This is a required, fixed length field when used for the set- and test password operations. If the password is entered as an ASCII string, it must be null (00h) terminated and 00h padded if the string is shorter than 20 bytes. This field is not needed if the operation is disable user or enable user. If this field is present for those operations, the MC ignores the data.		
Response data byte number	Data field		
1	Completion code. Generic, plus the command-specific completion codes:		
	80h =	Mandatory. Password test failed. Password size is correct, but the password data does not match the stored value.	
	81h =	Mandatory. Password test failed. Wrong password size was used.	

¹ The same user name can be used with different passwords on different channels. The MC scans user names until it finds the first one that is enabled for a particular channel. Thus, it is possible for a MC implementation to be configured to allow a 20-byte password on one channel, and a 16-byte password on another channel for the same user name. This requires multiple user entries.

RMCP+ support and payload commands

This sections list the commands associated with discovering, enabling, and activating payloads under IPMI v2.0/RMCP+ as well as updates and additions to IPMI commands to support IPMI v2.0/RMCP+ sessions, authentication, and configuration.

NOTE: The following commands remain available for payloads if IPMI messaging payload type is disabled:

- Deactivate payload
- Suspend/resume payload encryption (as defined for given payload)
- Get payload activation status
- Get channel payload version command
- Set session privilege level
- Close session

Table 43 (page 71) defines the payload type numbers and ranges for Payload Type Handles.

Table 43 Payload type numbers

Number	Type	Major format version	Minor format version
<i>Standard Payload Types</i>			
0h	IPMI Message	1h	0h
1h	SOL (serial over LAN)	1h	0h
<i>Session Setup Payload Types</i>			
10h	RMCP+ Open Session Request	1h	0h
11h	RMCP+ Open Session Response	1h	0h
12h	RAKP Message 1	1h	0h
13h	RAKP Message 2	1h	0h
14h	RAKP Message 3	1h	0h
15h	RAKP Message 4	1h	0h

Activate payload command

This command is available to the MC.

This command is used for activating and deactivating a payload type under a given IPMI session. The ability to execute this command is determined via the user's privileges as assigned via the `set user payload access` command.

The `activate payload` command may return a port number that is separate from the port number for the session that the command was issued under. In this case, the remote console must establish a session on the port number that the `activate session` command returned. The remote console must then issue the `activate payload` command on that port number in order to actually activate the payload. It is possible that the remote console already had a session active on the given port number. If the privileges associated with that session are sufficient (this is typically the case unless the remote console activated the session at a privilege level that was lower than the maximum level for the user) the remote console can re-use the existing session and just use the `activate payload` command to activate the new payload type.

BMCs may have limited resources for handling multiple sessions. It is highly recommended that a remote console avoids creating multiple sessions and shares sessions for multiple payloads whenever possible.

The `activate payload` command is only accepted over a channel on which payloads can be activated. For example, the `activate payload` command cannot be executed from the IPMB.

Table 44 Activate payload command request and response data

Request data byte number	Data field
1	<ul style="list-style-type: none"> • [7:6] — Reserved • [5:0] — Payload type. IPMI message payloads do not need to be explicitly activated. A payload that is required to be launched over a different port than that used to establish the initial IPMI session is only required to support the IPMI commands needed by the particular payload type.
2	Payload instance <ul style="list-style-type: none"> • [7:4] — Reserved • [3:0] — Payload instance. 1-based. 0h = reserved.
3:6	<p>Auxiliary request data. Additional payload-specific parameters to configure behavior of the payload when it becomes activated. Ignored if no auxiliary data is specified for a given payload type.</p> <p>For payload type = SOL:</p> <ul style="list-style-type: none"> • Byte 1 <ul style="list-style-type: none"> ◦ [7] — Encryption activation. The encryption algorithms specified in this document must be used with authentication. The MC returns an error completion code if an attempt is made to activate encryption without also activating authentication. <ul style="list-style-type: none"> – 1b: Activate payload with encryption. All SOL payload data from the MC is encrypted, if encryption was negotiated at the time of session activation. – 0b: Activate payload without encryption. MC sends all SOL payload data unencrypted, if that option is allowed. (An SOL configuration parameter allows a system to be configured to require encryption for all SOL transfers). ◦ [6] — Authentication activation. <ul style="list-style-type: none"> – 1b: Activate payload with authentication. All SOL payload data from the MC is authenticated, if authentication was negotiated at the time of session activation – 0b: Activate payload without authentication. MC sends all SOL payload data unauthenticated, if that option is allowed. (An SOL configuration parameter allows a system to be configured to require authentication for all SOL transfers). ◦ [5] — Test mode (optional). Enables DCD and DSR to be manually controlled by the remote console and the reporting of RTS and DTR state via the SOL operation/status byte. This can be used to facilitate software testing of the 16550 UART interface. <ul style="list-style-type: none"> – 1b = Activate test mode. If test mode is not supported, bit [0] of the auxiliary response data will be returned as 0b. – 0b = Deactivate test mode. ◦ [4] — Reserved ◦ [3:2] — Shared serial alert behavior. The following settings determine what happens to serial alerts if IPMI over serial and SOL are sharing the same baseboard serial controller. <ul style="list-style-type: none"> – 11b: Reserved – 10b: Serial/modem alerts succeed while SOL active. – 01b: Serial/modem alerts deferred while SOL active. – 00b: Serial/modem alerts fail while SOL active. ◦ [1] — SOL startup handshake <ul style="list-style-type: none"> – 0b: MC asserts CTS and DCD/DSR to baseboard upon activation. – 1b: CTS and DCD/DSR remain deasserted after activation. Remote console must send an SOL payload packet with control field settings to assert CTS and DCD/DSR. (This enables

Table 44 Activate payload command request and response data *(continued)*

	<p>the remote console to first alter volatile configuration settings before hardware handshake is released).</p> <ul style="list-style-type: none"> ◦ [0] — Reserved • Byte 2:4 — Reserved, write a 00h
Response data byte number	Data field
1	<p>Completion code. Generic plus the command-specific completion codes: (An error completion code should be returned if the payload type in the request is set to IPMI Message (0h)).</p> <ul style="list-style-type: none"> • 80h: Payload already active on another session (required). This will be returned any time an attempt is made to activate a payload type when that type is already activated for another session, and when the MC only supports one instance of that payload type running at a time. • 81h: Payload type is disabled (optional). Given payload type is not configured to be enabled for activation. • 82h: Payload activation limit reached. Cannot activate given payload type because the maximum number of simultaneous instances of that payload type are already running. • 83h: Cannot activate payload with encryption. • 84h: Cannot activate payload without encryption. MC requires encryption for all payloads for given privilege level.
2:5	<p>Auxiliary response data. LS-byte first. For payload = SOL:</p> <ul style="list-style-type: none"> • [31:1] — Reserved. Return as 0s. • [0] <ul style="list-style-type: none"> ◦ 0b = Test mode not supported / enabled. ◦ 1b = Test mode enabled.
6:7	Inbound payload size. Maximum size of payload data field from remote console to MC. Excludes size of confidentiality header and trailer fields, if any. 1-based.
8:9	Outbound payload size. Maximum size of payload data field from MC to remote console. Excludes size of confidentiality header and trailer fields, if any. 1-based.
10:11	<p>Payload UDP port number. UDP port number through which the payload can be transferred. If the port number is the same as the port that was used to establish the IPMI session, then SOL payload transfers are now available under that IPMI session on that port. Otherwise, the remote console needs to establish a separate IPMI session to the specified port number using the same IP address, username and password/key information that was used to establish the IPMI session. SOL payload transfers are then available over that session.</p> <p>If the remote console already has an IPMI session established on that port for a different payload type, the SOL payload type is also available over that session - provided that the session was established at a privilege level that matches the privilege level and authentication required for SOL. Otherwise, the remote console needs to close that session and reestablish it at the necessary privilege level.</p>
12:13	Payload VLAN number - FFFFh if VLAN addressing is not used.

Deactivate payload command

This command is available to the MC.

This command is used to terminate use of a given payload on an IPMI session. This type of traffic then becomes freed for activation by another session, or for possible re-activation under the present session. The `deactivate payload` command does not cause the session to be terminated. The `close session` command should be used for that purpose. A remote console application does not need to explicitly deactivate payload(s) before terminating a session. When a session terminates, all payloads that were active under that session are automatically deactivated by the MC.

Table 45 Deactivate payload command request and response data

Request data byte number	Data field
1	<ul style="list-style-type: none"> • [7:6] — Reserved • [5:0] — Payload type.
2	Payload instance <ul style="list-style-type: none"> • [7:4] — Reserved • [3:0] — Payload instance. 1-based. 0h = reserved.
3:6	Payload auxiliary data. Additional parameters to configure behavior of the payload when it becomes deactivated. Ignored if no auxiliary data is specified for given payload type. For payload type = SOL, (no auxiliary data) write as 0000_0000h:
Response data byte number	Data field
1	Completion code. Generic plus the command-specific completion codes: (An error completion code should be returned if the payload type in the request is set to “IPMI Message” (0h)). <ul style="list-style-type: none"> • 80h: Payload already deactivated. • 81h: Payload type is disabled (optional). Given payload type is not configured to be enabled for activation.

Suspend/resume payload encryption command

This command enables a remote console to control whether payload data from the MC is sent encrypted or not. Since encryption can be a significant burden on software, this command provides a mechanism to allow higher performance by operating without encryption and only activating encryption when it is required for data confidentiality. The command can also trigger a regeneration of the encryption Initialization Vector and re-initialization of the encryption state machine for algorithms such as xRC4 that use the same initialization vector for multiple packets.

The extent at which this command can control encryption of data from the MC is dependent on the payload definition. Some payload definitions may use a mix of encrypted and unencrypted payload data transfers. For example, a payload may implement a ‘request/response’ protocol, where the MC would return an encrypted or unencrypted response based on whether the request from the remote console was encrypted or unencrypted. In this case, the command may only affect data that is autonomously generated by the MC. Other payload definitions may just use whatever encryption the session was activated with, and offer no ‘run-time’ control of encryption/decryption, while other payload definitions may be ‘stream based’ where it is desirable for the remote console to be able to select when payload data is from the MC is encrypted or not.

The Suspend/Resume Payload Encryption command is only accepted from the channel that the payload was activated on.

Table 46 Payload-specific encryption behavior

Payload Type = IPMI Messaging
<ul style="list-style-type: none"> • Encrypted requests from the remote console will get encrypted responses from the MC. • The Suspend/Resume Payload Encryption command controls whether asynchronous (unrequested) messages from the MC are encrypted or not. • PET Traps (which are actually separate from IPMI Messaging) are always sent unencrypted.

Table 46 Payload-specific encryption behavior *(continued)*

Payload Type = SOL	
<ul style="list-style-type: none"> • The SOL configuration parameters allow configuring the system to require that SOL data be encrypted. • The MC will transmit SOL payload data according to encryption settings that were selected when the payload was activated unless over-ridden by SOL configuration parameters. • The Suspend/Resume Payload Encryption command controls whether SOL Payload data is encrypted or not. 	

Table 47 Suspend/resume payload command request and response data

IPMI request data byte number	Data field
1	[7:6] - reserved [5:0] - payload type (See Table 13-16, Payload Type Numbers)
2	Payload Instance [7:4] - reserved [3:0] - payload instance. 1-based. 0h = reserved.
3	[7:2] - reserved [4:0] - Operation <ul style="list-style-type: none"> • 2h = Regenerate initialization vector. For xRC4 encryption, this causes the MC to reinitialize the xRC4 state machine, reset the data offset, and deliver a new Initialization Vector value in the next encrypted packet it sends to the remote console. Because of processing delays and potential tasks in progress, the remote console may receive additional packets from the MC that are encrypted using the prior Initialization Vector before getting packets that use the new IV. • 1h = Resume/Start encryption on all transfers of specified payload data from the MC. • 0h = Suspend encryption on all transfers of specified payload messages from the MC.
IPMI response data byte number	Data field
1	Completion Code Generic plus the following command-specific completion codes: <ul style="list-style-type: none"> • 80h: Operation not supported for given payload type. • 81h: Operation not allowed under present configuration for given payload type. • 82h: Encryption is not available for session that payload type is active under. • 83h: The payload instance is not presently active.

Set channel security keys command

The Set Channel Security Keys command provides a standardized interface for initializing system unique keys that are used for the pseudo-random number generator key (KR) and the key-generation key (KG) used for RMCP+. Implementing the ability to set Kr is optional. The command is provided mainly to offer a common interface for BMCs that are not pre-configured with a KR values, or which may need their KR values to be restored if they are lost due to a data corruption or firmware update. The command includes a mechanism that allows specified keys to be “locked”. Once locked, the key value cannot be read back or rewritten via standard IPMI commands. It is possible, however, that a firmware update or re- installation procedure may cause the keys to be cleared or unlocked. Software utilities responsible for MC initial installation and setup should check to see whether keys have been locked and if not, should initialize them appropriately and lock them.

If this command is not supported, it indicates that the keys are either permanently pre-configured, or that they are only configurable via an OEM/MC-specific mechanism.

Request data byte number	Data field
1	<p>Channel Number</p> <p>[7:4] - reserved</p> <p>[3:0] - Channel Number</p> <p>NOTE: This command only applies to channels that support RMCP+, if the channel does not support RMCP+ the command will return an error completion code.</p>
2	<p>Operation</p> <p>[7:2] - reserved</p> <p>[1:0] - Operation</p> <ul style="list-style-type: none"> 00b = read key MC returns value of specified key, provided key has not yet been locked. Some BMCs may allow the key to be re-written if it does not match the expected value. Other BMCs may only allow one 'set' operation. If the key value has not yet been initialized, the MC will return 0's for the key value. Utility software responsible for MC installation and initial setup can use this operation to also check to see whether keys have been initialized and locked. 01b = set key MC writes given key value to non-volatile storage. 10b = lock key MC locks out modification or reading the key value. Once a key has been locked, it is not cannot be rewritten or read via IPMI specified commands. 11b = reserved
3	<p>Key ID</p> <p>[7:0] - key ID.</p> <ul style="list-style-type: none"> 00h = RMCP+ "KR" key (20 bytes). The "KR" key is used as a unique value for random number generation. Note: A MC implementation is allowed to share a single KR value across all channels. A utility can set KR and lock it for one channel, and then verify it has been set and locked for any other channels by using this command to read the key from other channels and checking the 'lock status' field for each channel to see if it matches and is locked. 01h = RMCP+ "KG" key (20 bytes). "KG" key acts as a value that is used for key exchange for the overall channel. This key is cannot be locked, to ensure a password/key configuration utility can set its value. This value is used in conjunction with the user key values (passwords) in RAKP-HMAC- SHA1 and RAKP-HMAC-MD5 authentication. I.e. the remote console needs to have a-priori knowledge of both this key value and the user password setting, in order to establish a session. KG must be individually settable on each channel that supports RMCP+. All other = reserved
(4:M)	<p>Key value. Value for specified key. Used for "set" Operation only. Otherwise, this field is not used in the request. The MC will ignore any bytes following the 'Key ID' byte.</p>
Response data byte number	Data field
1	<p>Completion Code. Generic, plus following command-specific completion codes:</p> <ul style="list-style-type: none"> 80h = Cannot perform set / confirm. Key is locked (mandatory) 81h = insufficient key bytes 82h = too many key bytes 83h = key value does not meet criteria for specified type of key 84h = KR is not used. MC uses a random number generation approach that does not require a KR value.

2	7:2 - reserved. 1:0 - lock status <ul style="list-style-type: none"> • 00b = key is not lockable. • 01b = key is locked. • 10b = key is unlocked. • 11b = reserved
(3:N)	Key value. The MC returns the specified key value when the Operation is set to "read key". Otherwise, the MC returns no additional bytes past the completion code.

Get system interface capabilities command

This command can be used to determine whether the SSIF supports multi-part transactions, and what size of IPMI messages can be transferred. The Get System Interface Capabilities command is mandatory for BMCs that implement multi-part writes or reads. Thus, software can assume that if the Get System Interface Capabilities command is not implemented, the interface only supports single-part writes and reads.

Request data byte number	Data field
1	System Interface Type [7:4] - reserved [3:0] - System Interface Type (For BT use the Get BT Interface Capabilities command) <ul style="list-style-type: none"> • 0h = SSIF • 1h = KCS • 2h = SMIC • all other = reserved
Response data byte number	Data field
1	Completion Code
2	Reserved. Returned as 00h.
For System Interface Type = SSIF:	
3	[7:6] - Transaction support <ul style="list-style-type: none"> • 00b = only single-part reads/writes supported. • 01b = multi-part reads/writes supported. Start and End transactions only. • 10b = multi-part reads/writes supported. Start, Middle, and End transactions supported. • 11b = reserved. [5:4] - reserved. [3] - PEC support. <ul style="list-style-type: none"> • 1b = implements PEC. MC will start using PEC in read transactions after it receives any SSIF write transaction that includes a valid PEC. The MC ceases using PEC if it receives an SSIF write transaction that does not include PEC. • 0b = does not support PEC. Note that a MC implementation may reject write transactions that include a PEC byte.

	[2:0] - SSIF Version <ul style="list-style-type: none"> 000b = version 1 (version defined in this specification).
4	Input message size in bytes. (1 based.) Number of bytes of IPMI message data that the MC can accept. This number does not include slave address, SMBus length, PEC, or SMBus CMD bytes, just the IPMI message data. A MC that just supports single-part writes would return 32 (20h) for this value. A MC that supports multi-part Start and End would return a value from 33 to 64. A MC that supports multi-part with Middle transactions would return a value from 65 to 255.
5	Output message size in bytes. (1 based.) Maximum number of bytes of IPMI message data that can be read from the MC. This number does not include slave address, SMBus length, PEC, SMBus CMD bytes, special bytes (such as the special bytes following the length byte in the multi-part read middle and end transactions) just the IPMI message data. A MC that just supports single-part reads would return 20h (32) for this value. A MC that supports multi-part Start and End would return a value from 33 to 62 (the reason this is 62 instead of 64 is that there are two special bytes after the length byte.) A MC that supports multi-part with Middle transactions would return a value from 63 to 255.
For System Interface Type = KCS or SMIC	
3	[7:3] - reserved [2:0] - System Interface Version <ul style="list-style-type: none"> 000b = version 1 (conformant with KCS or SMIC interface as defined in this specification).
4	Input maximum message size in bytes. (1 based.) Largest number of bytes that can be transferred in a KCS FFh means 255 or more.

Get payload activation status command

This command is available to the MC.

This command returns how many instances of a given payload type are presently activated, and how many total instances can be activated.

Table 48 Get payload activation status command request and response data

Request data byte number	Data field	
1	Payload type number - number of the standard payload type or OEM payload handle from which to retrieve status.	
Response data byte number	Data field	
1	Completion code	
2	Instance capacity	
	[7:4]	Reserved.
	[3:0]	Number of instances of a given payload type that can be simultaneously activated on MC. 1-based. 0h = reserved.
3	[7]	1b = Instance 8 is activated.
		0b = Instance 8 is deactivated.
	[6]	1b = Instance 7 is activated.
		0b = Instance 7 is deactivated.
	...	
	[0]	1b = Instance 1 is activated.

Table 48 Get payload activation status command request and response data *(continued)*

4	[7]	0b =	Instance 1 is deactivated.
		1b =	Instance 16 is activated.
		0b =	Instance 16 is deactivated.
	[6]	1b =	Instance 15 is activated.
		0b =	Instance 15 is deactivated.
	...		
	[0]	1b =	Instance 9 is activated.
		0b =	Instance 9 is deactivated.

Get payload instance info command

This command is available to the MC.

This command returns information about a specific instance of a payload type. It is primarily used by software that may want to negotiate with an application that is presently using the given payload type. It accomplishes this by using the session ID returned from this command with the `get session info` command to look up the addressing information for the party that activated the payload. The application may then use that information to establish a direct dialog with the application that presently owns the payload (this inter-application communication is not defined in the IPMI specifications).

Table 49 Get payload instance info command request and response data

Request data byte number	Data field
1	Payload type number - number of the standard payload type or OEM payload handle from which to retrieve status.
2	Payload instance. 1-based. 0h = reserved.
Response data byte number	Data field
1	Completion code. An error completion code should be returned if the payload type in the request is set to IPMI message (0h) .
2:5	Session ID - ID of session on which the instance is presently activated. (The managed system session ID that the MC generated when the session was activated). 00_00_00_00h if the given instance is not activated. Remote software can use this information with the <code>get session info</code> command to identify the remote console that presently is using a given payload type.
6:13	Payload-specific information (8-bytes) For payload type = SOL: <ul style="list-style-type: none"> Byte 1: Port number, a number representing the system serial port that is being redirected. 1-based. 0h = unspecified. Used when more than one port can be redirected on a system. Byte 2: 8 = reserved.

Set user payload access command

This command is available to the MC.

This command controls whether the specified user has the ability to activate the specified payload type on the given channel. The command uses bitfields to allow a configuration utility to use a single command to set enable/disable multiple payloads at a time. Standard payloads are set separately from OEM payload enables. The command would be issued at least once with standard

payloads selected to set the configuration for standard payloads, and then at least once with OEM payloads selected to set the configuration for OEM payloads.

Table 50 Set user payload access command request and response data

Request data byte number	Data field
1	Channel number <ul style="list-style-type: none"> [7:4] — Reserved [3:0] — Channel number
2	[7:6] - Operation <ul style="list-style-type: none"> 00b = Enable. Writing a "1b" to enable/disable bit enables the corresponding payload. Writing "0b" to bit causes no change to enabled/disabled state. 01b = Disable. Writing a "1b" to bit disables the corresponding payload. Writing "0b" to bit causes no change to enabled/disabled state. 10b, 11b = Reserved [5:0] — User ID. 000000b = reserved.
3	Standard payload enables 1 <ul style="list-style-type: none"> [7:2] — Reserved for standard payloads 2-7 enable/disable bits. [1] — Standard payload 1 (SOL) enable/disable [0] — Reserved. IPMI messaging is enabled/disabled for users via the <code>set user access</code> command.
4	Standard payload enables 2 - reserved
5	OEM payload enables 1 <ul style="list-style-type: none"> [7] - OEM payload 7 enable/disable [6] - OEM payload 6 enable/disable [5] - OEM payload 5 enable/disable [4] - OEM payload 4 enable/disable [3] - OEM payload 3 enable/disable [2] - OEM payload 2 enable/disable [1] - OEM payload 1 enable/disable [0] - OEM payload 0 enable/disable
6	OEM payload enables 2 - reserved
Response data byte number	Data field
1	Completion code. An implementation will not return an error completion code if the user access level is set higher than the privilege limit for a given channel. If it is desired to bring attention to this condition, it is up to software to check the channel privilege limits set using the <code>set channel access</code> command and provide notification of any mismatch.

Get user payload access command

This command is available to the MC.

The `get user payload access` command returns the user payload enable settings that were set using the `set user payload access` command.

Table 51 Get user payload access command request and response data

Request data byte number	Data field
1	Channel number <ul style="list-style-type: none"> [7:4] — Reserved [3:0] — Channel number
2	User ID <ul style="list-style-type: none"> [7:6] — Reserved [5:0] - User ID. 000000b = reserved
Response data byte number	Data field
1	Completion code
2	Standard payload enables 1 <ul style="list-style-type: none"> [7:2] — Reserved for standard payloads 2-7 enabled/disabled state. [1] <ul style="list-style-type: none"> 1b = Standard payload 1 enabled (SOL) 0b = Standard payload 1 disabled [0] — Reserved.
3	Standard payload enables 2 - reserved
4	OEM payload enables 1. For each bit: 1b = payload enabled, 0b = payload disabled. <ul style="list-style-type: none"> [7] - OEM payload 7 enabled/disabled [6] - OEM payload 6 enabled/disabled [5] - OEM payload 5 enabled/disabled [4] - OEM payload 4 enabled/disabled [3] - OEM payload 3 enabled/disabled [2] - OEM payload 2 enabled/disabled [1] - OEM payload 1 enabled/disabled [0] - OEM payload 0 enabled/disabled
5	OEM payload enables 2 - reserved

Get channel payload support command

This command is available to the MC.

This command enables local and remote console software to determine what payloads are enabled on the given MC. The command returns a bitfield indicating which payload type numbers can be activated on the given channel.

Table 52 Get channel payload support command request and response data

Request data byte number	Data field
1	Channel number <ul style="list-style-type: none"> [7:4] — Reserved [3:0] — Channel number
Response data byte number	Data field

Table 52 Get channel payload support command request and response data *(continued)*

1	Completion code
2	<ul style="list-style-type: none"> • [7] = Standard payload type #7 supported • ... • [0] = Standard payload type #0 supported
3	<ul style="list-style-type: none"> • [7] = Standard payload type #15 (0Fh) supported • ... • [0] = Standard payload type #8 supported
4	<ul style="list-style-type: none"> • [7] = Session setup payload type #7 supported • ... • [0] = Session setup payload type #0 supported
5	<ul style="list-style-type: none"> • [7] = Session setup payload type #15 (0Fh) supported • ... • [0] = Session setup payload type #8 supported
6	<ul style="list-style-type: none"> • [7] = Payload type 27h (OEM7) used • ... • [0] = Payload type 20h (OEM0) used
7	<ul style="list-style-type: none"> • [7] = Payload type 2Fh (OEM15) used • ... • [0] = Payload type 28h (OEM8) used
8:9	Reserved. Return as 0000h

Get channel payload version command

This command is available to the MC.

This command returns version information for the given payload type. The version number has major and minor parts. The major part of the version should only increment when there are significant changes to the payload format, commands, or payload-specific protocols that break backward compatibility with earlier versions. The minor part of the version increments when there are extensions to the payload format that are significant but are backwards compatible with earlier versions under the same major version number. An example of a major change would be a change to the payload activation process that would prevent earlier applications from activating the given payload type. An example of a minor format version change would be the definition of commands for new functions that did not exist under the previous format, but if unused, do not interfere with the operation of older applications.

Table 53 Get channel payload version command request and response data

Request data byte number	Data field
1	Channel number <ul style="list-style-type: none"> • [7:4] — Reserved • [3:0] — Channel number
2	Payload type number/payload type handle - number of the standard payload type or OEM payload handle for which to retrieve status. See Table 43 (page 71) .

Table 53 Get channel payload version command request and response data *(continued)*

Response data byte number	Data field
1	Completion code. Generic plus command-specific completion code: 80h — Payload type not available on given channel.
2	Format version <ul style="list-style-type: none"> • [7:4] - Major format version. BCD encoded (0 to 9). • [3:0] - Minor format version. BCD encoded (0 to 9). Software should present version data to the user in the format “major.minor”. For example, 10h → 1.0. The format version for the SOL payload implemented per this specification is 1.0 (10h).

IPMI LAN Device Commands

This section defines the configuration and control commands that are specific to LAN channels. None of the commands in the following table are required unless a LAN channel is implemented. See [Table 125 \(page 160\)](#) for the specification of the Network Function and Command (CMD) values and privilege levels for these commands.

Set LAN configuration parameters command

This command is used for setting parameters such as the network addressing information required for IPMI LAN-operation.

Table 54 Set LAN configuration parameters request and response data

Request data byte number	Data field
1	Channel number <ul style="list-style-type: none"> • [7:4] — Reserved • [3:0] — Channel number
2	Parameter selector
3:N	Configuration parameter data, per the table.
Response data byte number	Data field
1	Completion code. <ul style="list-style-type: none"> • 80h = parameter not supported. • 81h = attempt to set the ‘set in progress’ value (in parameter #0) when not in the ‘set complete’ state. (This completion code provides a way to recognize that another party has already ‘claimed’ the parameters.) • 82h = attempt to write read-only parameter. • 83h = attempt to read write-only parameter.

Get LAN configuration parameters command

This command is used for retrieving the configuration parameters from the `set LAN configuration parameters` command.

Table 55 Get LAN configuration parameters request and response data

Request data byte number	Data field
1	[7] <ul style="list-style-type: none"> 0b = Get parameter 1b = Get parameter revision only [6:4] - Reserved [3:0] - Channel number
2	Parameter selector
3	Set Selector. Selects a given set of parameters under a given Parameter selector value. 00h if parameter doesn't use a Set Selector.
4	Block Selector (00h if parameter does not require a block number)
Response data byte number	Data field
1	Completion Code. Generic codes, plus following command-specific completion code(s): 80h = parameter not supported.
2	[7:0] - Parameter revision. Format: MSN = Present revision. LSN = Oldest revision with which the parameter is backward compatible. 11h for parameters in this specification.
The following data bytes are not returned when the 'get parameter revision only' bit is 1b.	
3:N	Configuration parameter data, per Table 56 (page 84) . If the rollback feature is implemented, the MC makes a copy of the existing parameters when the 'set in progress' state becomes asserted (See the Set In Progress parameter #0). While the 'set in progress' state is active, the MC will return data from this copy of the parameters, plus any uncommitted changes that were made to the data. Otherwise, the MC returns parameter data from non-volatile storage.

Table 56 LAN configuration parameters

Parameter	#	Parameter Data (non-volatile unless otherwise noted) ¹
Set In Progress (volatile)	0	<p><u>data 1</u> - This parameter is used to indicate when any of the following parameters are being updated, and when the updates are completed. The bit is primarily provided to alert software than some other software or utility is in the process of making changes to the data. An implementation can also elect to provide a 'rollback' feature that uses this information to decide whether to 'roll back' to the previous configuration information, or to accept the configuration change.</p> <p>If used, the roll back shall restore all parameters to their previous state. Otherwise, the change shall take effect when the write occurs.</p> <p>[7:2] - reserved [1:0] -</p> <ul style="list-style-type: none"> 00b = set complete. If a system reset or transition to powered down state occurs while 'set in progress' is active, the MC will go to the 'set complete' state. If rollback is implemented, going directly to 'set complete' without first doing a 'commit write' will cause any pending write data to be discarded. 01b = set in progress. This flag indicates that some utility or other software is presently doing writes to parameter data. It is a notification flag only, it is not a resource lock. The MC does not provide any interlock mechanism that would prevent other software from writing parameter data while. 10b = commit write (optional). This is only used if a rollback is implemented. The MC will save the data that has been written since the last time the 'set in

Table 56 LAN configuration parameters *(continued)*

Parameter	#	Parameter Data (non-volatile unless otherwise noted) ¹
		<p>progress' and then go to the 'set in progress' state. An error completion code will be returned if this option is not supported.</p> <ul style="list-style-type: none"> 11b = reserved
Authentication Type Support (Read Only)	1	<p>This 'read only' field returns which possible Authentication Types (algorithms) can be enabled for the given channel. The following Authentication Type Enables parameter selects which Authentication Types are available when activating a session for a particular maximum privilege level.</p> <p>[7:6] -reserved</p> <p>[5:0] -Authentication type(s) enabled for this channel (bitfield): All bits:</p> <ul style="list-style-type: none"> 1b = supported 0b = authentication type not available for use. <p>[5] - OEM proprietary (per OEM identified by the IANA OEM ID in the RMCP Ping Response)</p> <p>[4] - straight password / key</p> <p>[3] - reserved</p> <p>[2] - MD5</p> <p>[1] - MD2</p> <p>[0] - none</p>
Authentication Type Enables	2	<p>This field is used to configure which Authentication Types are available for use when a remote console activates an IPMI messaging connection to the MC for a given requested maximum privilege level. Once the session has been activated, the accepted authentication type will be the only one used for authenticated packets, regardless of the present operating privilege level, or the privilege level associated with the command.</p> <p>Depending on configuration of per-message and user-level authentication disables, unauthenticated packets (authentication type = none) may also be accepted. The MC makes no attempt to check or ensure that stricter authentication types are associated with higher requested maximum privilege levels. E.g. it is possible to configure the MC so activating a session with a maximum privilege level of 'User' requires MD5 while 'Admin' requires 'none'.</p> <p>NOTE: An implementation that has fixed privilege and authentication type assignments, in which case this parameter can be implemented as Read Only. It is recommended that an implementation that implements a subset of the possible authentication types returns a CCh error completion code if an attempt is made to select an unsupported authentication type.</p> <ul style="list-style-type: none"> byte 1: Authentication Types returned for maximum requested privilege = Callback level. <ul style="list-style-type: none"> [7:6] -reserved [5:0] -Authentication type(s) enabled for this channel (bitfield): All bits: <ul style="list-style-type: none"> 1b = authentication type enabled for use at given privilege level 0b = authentication type not available for use at given privilege level. [5] - OEM proprietary (per OEM identified by the IANA OEM ID in the RMCP Ping Response) [4] - straight password / key [3] - reserved [2] - MD5

Table 56 LAN configuration parameters *(continued)*

Parameter	#	Parameter Data (non-volatile unless otherwise noted) ¹
		<ul style="list-style-type: none"> ◦ [1] - MD2 ◦ [0] - none • <u>byte 2</u>: Authentication Type(s) for maximum privilege = User level (format follows byte 1) • <u>byte 3</u>: Authentication Type (s) for maximum privilege = Operator level (format follows byte 1) • <u>byte 4</u>: Authentication Type (s) for maximum privilege = Administrator level (format follows byte 1) • <u>byte 5</u>: Authentication Type (s) for maximum privilege = OEM level (format follows byte 1)
IP Address	3	<u>data 1:4</u> - IP Address MS-byte first.
IP Address Source	4	<u>data 1</u> [7:4] -reserved [3:0] -address source <ul style="list-style-type: none"> • 0h = unspecified • 1h = static address (manually configured) • 2h = address obtained by MC running DHCP • 3h = address loaded by BIOS or system software • 4h = address obtained by MC running other address assignment protocol
MAC Address (can be Read Only)	5	<u>data 1:6</u> - MAC Address for messages transmitted from MC. MS-byte first. An implementation can either allow this parameter to be settable, or it can be implemented as Read Only.
Subnet Mask	6	<u>data 1:4</u> - Subnet Mask. MS-byte first.
IPv4 Header Parameters	7	<ul style="list-style-type: none"> • <u>data 1</u> - Time-to-live. 1-based. (Default = 40h) Value for time-to-live parameter in IP Header for RMCP packets and PET Traps transmitted from this channel. • <u>data 2</u> <ul style="list-style-type: none"> ◦ [7:5] - Flags. Sets value of bit 1 in the Flags field in the IP Header for packets transmitted by this channel. (Default = 010b "don't fragment") ◦ [4:0] - reserved • <u>data 3</u> <ul style="list-style-type: none"> ◦ [7:5] - Precedence (Default = 000b) ◦ [4:1] - Type of Service (Default = 1000b, "minimize delay") [0] - reserved
Primary RMCP Port Number (optional)	8	<u>data 1:2</u> - Primary RMCP Port Number, LSByte first. Default = 26Fh (RMCP 'Aux Bus Shunt' port)
Secondary RMCP Port Number (optional)	9	<u>data 1:2</u> - Secondary Port Number, LSByte first. Default = 298h (RMCP 'Secure Aux Bus' port)

Table 56 LAN configuration parameters *(continued)*

Parameter	#	Parameter Data (non-volatile unless otherwise noted) ¹
MC-generated ARP control (optional ²)	10	<p><u>data 1</u> - MC-generated ARP control. Note: the individual capabilities for MC-generated ARP responses and MC-generated Gratuitous ARPs are individually optional. The MC should return an error completion code if an attempt is made to enable an unsupported capability.</p> <p>[7:2] - reserved</p> <p>[1] -</p> <ul style="list-style-type: none"> 1b = enable MC-generated ARP responses 0b = disable MC-generated ARP responses <p>[0] -</p> <ul style="list-style-type: none"> 1b = enable MC-generated Gratuitous ARPs 0b = disable MC-generated Gratuitous ARPs
Gratuitous ARP interval (optional)	11	<p><u>data 1</u> -</p> <ul style="list-style-type: none"> Gratuitous ARP interval Gratuitous ARP interval in 500 millisecond increments. 0-based. Interval accuracy is +/- 10%. If this configuration parameter is not implemented, gratuitous ARPs shall be issued at a rate of once every 2 seconds.
Default Gateway Address	12	<p><u>data 1:4</u> - IP Address</p> <p>MS-byte first. This is the address of the gateway (router) used when the MC sends a message or alert to a party on a different subnet than the one the MC is on.</p>
Default Gateway MAC Address	13	<p><u>data 1:6</u> - MAC Address. MS-byte first.</p>
Backup Gateway Address	14	<p><u>data 1:4</u> - IP Address</p> <p>MS-byte first. This is the address of an alternate gateway (router) that can be selected when a sending a LAN Alert.</p>
Backup Gateway MAC Address	15	<p><u>data 1:6</u> - MAC Address. MS-byte first.</p>
Community String	16	<p><u>data 1:18</u> - Community String</p> <p>Default = 'public'. Used to fill in the 'Community String' field in a PET Trap. This string may optionally be used to hold a vendor-specific string that is used to provide the network name identity of the system that generated the event. Printable ASCII string-. If a full 18 non-null characters are provided, the last character does not need to be a null. 18 characters must be written when setting this parameter, and 18 will be returned when this parameter is read.</p> <p>The null character, and any following characters, will be ignored when the Community String parameter is placed into the PET. The MC will return whatever characters were written. In other words, it will not set bytes following the null to any particular value.</p>
Number of Destinations (Read Only)	17	<p><u>data 1</u> - Number of LAN Alert Destinations supported on this channel. (Read Only). At least one set of non-volatile destination information is required if LAN alerting is supported. Additional non-volatile destination parameters can optionally be provided for supporting an alert 'call down' list policy. A maximum of fifteen (1h to Fh) non-volatile destinations are supported in this specification. Destination 0 is always present as a volatile destination that is used with the Alert Immediate command.</p> <p>[7:4] - reserved.</p> <p>[3:0] - Number LAN Destinations. A count of 0h indicates LAN Alerting is not supported.</p>

Table 56 LAN configuration parameters *(continued)*

Parameter	#	Parameter Data (non-volatile unless otherwise noted) ¹
Destination Type (volatile / non-volatile - see description)	18	<p>Sets the type of LAN Alert associated with the given destination. This parameter is not present if the Number of Destinations parameter is 0.</p> <ul style="list-style-type: none"> • <u>data 1</u> - Set Selector = Destination selector, 0 based. <ul style="list-style-type: none"> ◦ [7:4] -reserved ◦ [3:0] -Destination selector. Destination 0 is always present as a volatile destination that is used with the Alert Immediate command. • <u>data 2</u> - Destination Type <ul style="list-style-type: none"> ◦ [7] - Alert Acknowledge. <ul style="list-style-type: none"> - 0b = Unacknowledged. Alert is assumed successful if transmission occurs without error. This value is also used with Callback numbers. - 1b = Acknowledged. Alert is assumed successful only if acknowledged is returned. Note, some alert types, such as Dial Page, do not support an acknowledge. ◦ [6:3] -reserved ◦ [2:0] -Destination Type <ul style="list-style-type: none"> - 000b = PET Trap destination - 001b - 101b = reserved - 110b = OEM 1 - 111b = OEM 2 • <u>data 3</u> - Alert Acknowledge Timeout / Retry Interval, in seconds, 0-based (i.e. minimum timeout = 1 second). <p>This value sets the timeout waiting for an acknowledge, or the time between automatic retries depending on whether the alert is acknowledge or not. Recommended factory default = 3 seconds. Value is ignored if alert type does not support acknowledge, or if the Alert Acknowledge bit (above) is 0b.</p> • <u>data 4</u> - Retries <ul style="list-style-type: none"> ◦ [7:4] - Reserved ◦ [3] - Reserved ◦ [2:0] - Number of times to retry alert to given destination. 0 = no retries (alert is only sent once). If the alert is acknowledged (Alert Acknowledge bit = 1b) the alert will only be retried if a timeout occurs waiting for the acknowledge. Otherwise, this value selects the number of times an unacknowledged alert will be sent out. The timeout interval or time between retries is set by the Alert Acknowledge Timeout / Retry Interval value (byte 3 of this parameter).

Table 56 LAN configuration parameters *(continued)*

Parameter	#	Parameter Data (non-volatile unless otherwise noted) ¹
Destination Addresses	19	<p>Sets/Gets the list of IP addresses that a LAN alert can be sent to. This parameter is not present if the Number of Destinations parameter is 0.</p> <ul style="list-style-type: none"> <u>data 1</u> - Set Selector = Destination Selector. <ul style="list-style-type: none"> [7:4] -reserved [3:0] -Destination selector. Destination 0 is always present as a volatile destination that is used with the Alert Immediate command. <u>data 2</u> - Address Format <ul style="list-style-type: none"> [7:4] - Address Format. 0h = IPv4 IP Address followed by DIX Ethernet/802.3 MAC Address [3:0] - Reserved <p>For Address Format = 0h:</p> <ul style="list-style-type: none"> <u>data 3</u> - Gateway selector <ul style="list-style-type: none"> [7:1] - Reserved [0] - <ul style="list-style-type: none"> 0b = use default gateway 1b = use backup gateway <u>data 4:7</u> - Alerting IP Address (MS-byte first) <u>data 8:13</u> - Alerting MAC Address (MS-byte first)
<p>Following parameters are introduced with IPMI v2.0 / RMCP+</p> <p>VLAN configuration can be used with IPMI v1.5 and IPMI v2.0 sessions. Parameters labeled "RMCP+" are specific to IPMI v2.0 implementations that implement IPMI v2.0 / RMCP+ sessions.</p>		
802.1q VLAN ID (12-bit)	20	<ul style="list-style-type: none"> <u>data 1</u> [7:0] - Least significant 8-bits of the VLAN ID. 00h if VLAN ID not used. <u>data 2</u> <ul style="list-style-type: none"> [7] - VLAN ID enable. <ul style="list-style-type: none"> 0b = disabled 1b = enabled. <p>If enabled, the MC will only accept packets for this channel if they have 802.1q fields and their VLAN ID matches the VLAN ID value given in this parameter.</p> [6:4] - Reserved [3:0] - Most significant four bits of the VLAN ID
802.1q VLAN Priority	21	<p><u>data 1</u> [7:3] - Reserved</p> <p>[2:0] - Value for Priority field of 802.1q fields. Ignored when VLAN ID enable is 0b (disabled) - See 802.1q VLAN ID parameter, above. Setting is network dependent. By default, this should be set to 000b.</p>
RMCP+ Messaging Cipher Suite Entry Support (Read Only)	22	<p>This parameter provides a count of the number (16 max.) of Cipher Suites available to be enabled for use with IPMI Messaging on the given channel.</p> <p>Software can find out what security algorithms are associated with given Cipher Suite ID by using the Get Channel Cipher Suites command. In addition, there are</p>

Table 56 LAN configuration parameters *(continued)*

Parameter	#	Parameter Data (non-volatile unless otherwise noted) ¹
		<p>Cipher Suite IDs assigned for standard Cipher Suites (see Table 22-19, Cipher Suite IDs)</p> <p><u>data 1</u></p> <p>[7:5] - reserved</p> <p>[4:0] - Cipher Suite Entry count. Number of Cipher Suite entries, 1-based, 10h max.</p>
RMCP+ Messaging Cipher Suite Entries (Read Only)	23	<p>This parameter contains zero to sixteen (16) bytes of Cipher Suite IDs for Cipher Suites that can be used for establishing an IPMI messaging session with the MC. The number of Cipher Suites that are supported is given in the preceding parameter.</p> <ul style="list-style-type: none"> • <u>data 1</u> - Reserved • <u>data 2</u> - Cipher Suite ID entry A. <u>data 3</u> - Cipher Suite ID entry B. • ... • <u>data 17</u> - Cipher Suite ID entry P.
RMCP+ Messaging Cipher Suite Privilege Levels	24	<p>This parameter allows the configuration of which privilege levels are associated with each Cipher Suite. The total number of nibbles supported (zero to sixteen) matches the number of fixed Cipher Suite IDs.</p> <ul style="list-style-type: none"> • <u>data 1</u> - Reserved • <u>data 2</u> - Maximum Privilege Level for 1st and 2nd Cipher Suites <ul style="list-style-type: none"> ◦ [7:4] - Maximum Privilege Level for 2nd Cipher Suite ◦ [3:0] - Maximum Privilege Level for 1st Cipher Suite <ul style="list-style-type: none"> - 0h = Unspecified (given Cipher Suite is unused) - 1h = Callback level - 2h = User level - 3h = Operator level - 4h = Administrator level - 5h = OEM Proprietary level • <u>data 3</u> - Maximum Privilege Level for 3rd and 4th Cipher Suites <u>data 4</u> - Maximum Privilege Level for 5th and 6th Cipher Suites • ... • <u>data 9</u> - Maximum Privilege Level for 15th and 16th Cipher Suites
Destination Address VLAN TAGs (can be READ ONLY, see description)	25	<p>Sets/Gets the VLAN IDs (if any) addresses that a LAN alert can be sent to. This parameter is not present if the Number of Destinations parameter is 0, or if the implementation does not support the use of VLAN IDs for alerts. Otherwise, the number of VLAN TAG entries matches the number of Alert Destinations.</p> <p>An implementation may only be able to send alerts using the same VLAN TAG configuration as specified by parameters 20 and 21, in which case this parameter is allowed to be READ ONLY, where data 3-4 reflects the settings of parameters 20 and 21, and data 2 [7:4] indicates that VLAN TAGs are being used for alerts. If the implementation does support configurable VLAN TAGs for alert destinations,</p>

Table 56 LAN configuration parameters *(continued)*

Parameter	#	Parameter Data (non-volatile unless otherwise noted) ¹
		<p>it must support configuring unique TAG information for all destinations on the given channel.</p> <ul style="list-style-type: none"> • <u>data 1</u> - Set Selector = Destination Selector. <ul style="list-style-type: none"> ◦ [7:4] -reserved ◦ [3:0] -Destination selector. Destination 0 is always present as a volatile destination that is used with the Alert Immediate command. • <u>data 2</u> - Address Format <ul style="list-style-type: none"> ◦ [7:4] - Address Format. <ul style="list-style-type: none"> – 0h = VLAN ID not used with this destination – 1h = 802.1q VLAN TAG ◦ [3:0] - Reserved <p>For Address Format = 1h:</p> <ul style="list-style-type: none"> • <u>data 3:-4</u> - VLAN TAG <ul style="list-style-type: none"> ◦ [7:0] - VLAN ID, least-significant byte ◦ [11:8] - VLAN ID, most-significant nibble ◦ [12] - CFI (Canonical Format Indicator. Set to 0b) [15:13] - User priority (000b, typical)
Bad Password Threshold (optional)	26	<p>Sets/Gets the Bad Password Threshold. If implemented and non-zero, this value determines the number of sequential bad passwords that will be allowed to be entered for the identified user before the user is automatically disabled from access on the channel.</p> <p>For example, a value of 3 indicates that 3 sequential attempts are allowed for the given username on the particular channel. If the password for the third attempt is not correct, the user will be disabled for the channel. If this value is zero (00h) then there is no limit on bad passwords.</p> <p>The effect of the disable is the same as if a Set User Access command were used to remove the user's access from the channel.</p> <p>Bad password attempts are tracked according to individual username on a per channel basis. (Thus, a given username may be disabled on one channel, but still enabled on another) Bad password attempts are not counted if integrity check or other session parameters, such as session ID, sequence number, etc. are invalid. That is, bad password attempts are not counted if there are any other errors that would have caused the login attempt to be rejected even if the password was valid. The count of bad password attempts is retained as long as the MC remains powered and is not reinitialized.</p> <p>Counting automatically starts over (is reset) under any one of the following conditions:</p> <ul style="list-style-type: none"> • A valid password is received on any of the allowed attempts b) the Attempt Count Reset Interval expires • The user is re-enabled using the Set User Access command • The user is automatically re-enabled when the User Lockout Interval expires. • The Bad Threshold number parameter value is re-written or changed

Table 56 LAN configuration parameters *(continued)*

Parameter	#	Parameter Data (non-volatile unless otherwise noted) ¹
		<p>The Set User Access command is used to re-enable the user for the Channel.</p> <ul style="list-style-type: none"> • byte 1 <ul style="list-style-type: none"> ◦ [7:1] - reserved ◦ [0] - <ul style="list-style-type: none"> – 0b = do not generate an event message when the user is disabled. – 1b = generate a Session Audit sensor "Invalid password disable" event message. byte 2 ◦ 7:0 - Bad Password Threshold number. • byte 3:4 <ul style="list-style-type: none"> ◦ 15:0 - Attempt Count Reset Interval. The interval, in tens of seconds, for which the accumulated count of bad password attempts is retained before being automatically reset to zero. The interval starts with the most recent bad password attempt for the given username on the channel. This interval is allowed to reset if a MC power cycles or re-initialization occurs while the interval is being counted. <p>0000h = Attempt Count Reset Interval is disabled. The count of bad password attempts is retained as long as the MC remains powered and is not reinitialized.</p> • byte 5:6 <ul style="list-style-type: none"> ◦ 15:0 - User Lockout Interval. The interval, in tens of seconds, that the user will remain disabled after being disabled because the Bad Password Threshold number was reached. The user is automatically re-enabled when the interval expires. Note that this requires the MC implementation to track that the user was disabled because of a Bad Password Threshold. This interval is allowed to be restarted if a MC power cycle or re-initialization occurs while the interval is being counted. Note that this requires an internal non-volatile setting to be maintained that tracks when a particular user has been temporarily disabled due to the Bad Password Threshold. This is required to distinguish a user that was disabled automatically from a user that is intentionally disabled using the Set User Access command. <p>0000h = User Lockout Interval is disabled. If a user was automatically disabled due to the Bad Password threshold, the user will remain disabled until re-enabled via the Set User Access command.</p>
OEM Parameters	192 : 255	This range is available for special OEM configuration parameters. The OEM is identified according to the Manufacturer ID field returned by the Get Device ID command.

¹ Choice of system manufacturing defaults is left to the system manufacturer unless otherwise specified.

² This configuration parameter must be supported if the controller autonomously issues gratuitous ARPs or ARP responses.

SOL commands

Set SOL configuration parameters command

This command is available to the MC.

This command is used for setting parameters such as the network addressing information required for SOL payload operation. Parameters can be volatile or non-volatile.

Table 57 Set SOL configuration parameters command request and response data

Request data byte number	Data field
1	<ul style="list-style-type: none"> [7:4] — Reserved [3:0] — Channel number
2	Parameter selector
3:N	Configuration parameter data. See Table 59 (page 94) .
Response data byte number	Data field
1	<p>Completion code.</p> <ul style="list-style-type: none"> 80h = Parameter not supported. 81h = Attempt to set the set in progress value (in parameter #0) when not in the set complete state. (This completion code provides a way to recognize that another party has already “claimed” the parameters). 82h = Attempt to write read-only parameter. 83h = Attempt to read write-only parameter.

Get SOL configuration parameters command

This command is available to the MC.

This command is used for retrieving the configuration parameters from the `set sol configuration parameters` command.

Table 58 Get SOL configuration parameters command request and response data

Request data byte number	Data field
1	<ul style="list-style-type: none"> [7] <ul style="list-style-type: none"> 0b = Get parameter 1b = Get parameter revision only [6:4] — Reserved [3:0] — Channel number
2	Parameter selector
3	Set selector. Selects a given set of parameters under a given parameter selector value. 00h if parameter does not use a set selector.
4	Block selector (00h if parameter does not require a block number).
Response data byte number	Data field
1	Completion code. Generic codes, plus command-specific completion code: 80h = parameter not supported.
	[7:0] - Parameter revision. Format: MSN = present revision. LSN = oldest revision parameter in which it is backward compatible. 11h for parameters in this specification.
The following data byte is not returned when the get parameter revision only bit is 1b.	
3:N	Configuration parameter data, per Table 59: “SOL configuration parameters” (page 94) If the rollback feature is implemented, the MC makes a copy of the existing parameters when the set in progress state becomes asserted. (See the set in progress parameter #0). While the set in progress state is active, the MC returns data from this copy of the parameters, plus any uncommitted changes that were made to the data. Otherwise, the MC returns parameter data from non-volatile storage.

Table 59 SOL configuration parameters

Parameter	#	Parameter Data (non-volatile unless otherwise noted) ¹	
Set In Progress (volatile)	0	<p>Data 1 - This parameter is used to indicate when any of the following parameters are being updated, and when the updates are completed. The bit is primarily provided to alert software that some other software or utility is in the process of making changes to the data.</p> <p>An implementation can also elect to provide a rollback feature that uses this information to decide whether to roll back to the previous configuration information, or to accept the configuration change.</p> <p>If used, the roll back restores all parameters to their previous state. Otherwise, the change takes effect when the write occurs.</p>	
		[7:2]	Reserved
		[1:0]	<p>00b = Set complete. If a system reset or transition to powered down state occurs while set in progress is active, the MC goes to the set complete state. If rollback is implemented, going directly to set complete without first doing a commit write causes any pending write data to be discarded.</p>
			<p>01b = Set in progress. This flag indicates that some utility or other software is presently doing writes to parameter data. It is a notification flag only, it is not a resource lock. The MC does not provide any interlock mechanism that would prevent other software from writing parameter data.</p>
			<p>10b = Commit write (optional). This is only used if a rollback is implemented. The MC saves the data that has been written since the last time it was set in progress and then goes to the set in progress state. An error completion code will be returned if this option is not supported.</p>
		11b =	Reserved
SOL enable	1	Byte 1:	
		[7:1]	Reserved
		[0]	SOL enable. This controls whether the SOL payload type can be activated. Whether an SOL stream can be established is also dependent on the access mode and authentication settings for the corresponding LAN channel. The enabled/disabled state and access mode settings for the serial/modem channel have no effect on SOL.
		1b =	Enable SOL payload
		0b =	Disable SOL payload
SOL authentication	2	Byte 1: SOL authentication enable	
		[7]	Force SOL payload encryption
		1b:	Force encryption. If the cipher suite for the session supports encryption, this setting forces the use of encryption for all SOL payload data.
		0b:	Encryption controlled by remote console. Whether SOL packets are encrypted or not is selected by the remote console at the time the payload is activated (using the activate payload command) and can be changed during operation via the suspend/resume payload encryption command.
		[6]	Force SOL payload authentication

Table 59 SOL configuration parameters *(continued)*

Parameter	#	Parameter Data (non-volatile unless otherwise noted) ¹	
		1b:	Force authentication. If the cipher suite for the session supports authentication, this setting forces the use of authentication on all SOL payload data.
		0b:	Authentication controlled by remote software. For the standard cipher suites, if encryption is used then authentication must also be used. Therefore, while encryption is being used, software is not be able to select using unauthenticated payloads.
		[5:4]	Reserved
		[3:0]	SOL privilege level. Sets the minimum operating privilege level that is required to be able to activate SOL using the <code>activate</code> payload command.
		0h:	Reserved
		1h:	Reserved
		2h:	User level
		3h:	Operator level
		4h:	Administrator level
		5h:	OEM proprietary level
		All other :	Reserved
Character accumulate interval & character send threshold	3	<ul style="list-style-type: none"> Byte 1: Character accumulate interval in 5 ms increments. 1-based. This sets the typical amount of time that the MC waits before transmitting a partial SOL character data packet. (Where a partial packet is defined as a packet that has fewer characters to transmit than the number of characters specified by the Send threshold parameter (see below). A packet is not sent. 00h = reserved Byte 2: Character send threshold. 1-based. The MC automatically sends an SOL character data packet containing this number of characters as soon as this number of characters (or greater) has been accepted from the baseboard serial controller into the MC. This provides a mechanism to tune the buffer to reduce latency to when the first characters are received after an idle interval. In the degenerate case, setting this value to 1 would cause the MC to send a packet as soon as the first character was received. This can be useful if the character accumulate interval is large. If the MC is waiting for an acknowledge from the previous packet, it ignores this threshold and continues to collect data until it has a full packet's worth. 	
SOL retry	4	<ul style="list-style-type: none"> Byte 1: Retry count <ul style="list-style-type: none"> [7:3] - Reserved [2:0] - Retry count. 1-based. 0 = no retries after packet is transmitted. Packet is dropped if no ACK/NACK received by the time retries expire. Byte 2: Retry interval. 1-based. Retry interval in 10 ms increments. Sets the time that the MC waits before the first retry and the time between retries when sending SOL packets to the remote console. <ul style="list-style-type: none"> 00h: Retries sent back-to-back 	
SOL non-volatile bit rate (non-volatile)	5	<p>This configuration parameter is not supported if the implementation does not have a MC serial controller that can be potentially configured.</p> <p>Serial communication with the MC when SOL is activated always occurs using 8 bits/character, no parity, 1 stop bit, and RTS/CTS (hardware) flow control.</p>	

Table 59 SOL configuration parameters *(continued)*

Parameter	#	Parameter Data (non-volatile unless otherwise noted) ¹
		<p>NOTE: If SOL is enabled for multiple LAN channels, the MC uses the serial communication settings for the channel over which the <code>activate sol</code> command was initially received. The settings for other channels are ignored.</p> <p>Data 1</p>
		[7:4] Reserved
		[3:0] Bit rate. 1-5h = reserved. Support for bit rates other than 19.2 kbps is optional. The MC must return an error completion if a requested bit rate is not supported. It is recommended that the parameter out-of-range (C9h) code be used for this situation.
		0h = Use setting presently configured for IPMI over serial channel. The setting is used even if the access mode for the serial channel is set to disabled. IPMI specification can allow more than one serial channel. If serial port sharing is not implemented, this value is reserved. 6h: 9600 bps.
		7h = 19.2 kbps
		8h = 38.4 kbps
		9h = 57.6 kbps
		Ah = 115.2 kbps
		All other = Reserved
SOL volatile bit rate (volatile)	6	Set volatile version of SOL serial settings. Data follows that for the SOL non-volatile bit rate parameter.
SOL payload channel (optional, read only)	7	This parameter indicates which IPMI channel is being used for the communication parameters (such as IP address, MAC address) for the SOL payload. Typically, these parameters come from the same channel that the <code>activate payload</code> command for SOL was accepted.
SOL payload port number (read only or read/write)	8	<p>This parameter is read/write when the implementation allows the port number over which the SOL payload can be activated to be configurable. Otherwise, it is a read only parameter.</p> <p>Data 1:2 - Primary RMCP port number, LS byte first.</p>
OEM parameters	192:255	This range is available for special OEM configuration parameters. The OEM is identified according to the manufacturer ID field returned by the <code>get device id</code> command.

¹ Choice of system manufacturing defaults is left to the system manufacturer unless otherwise specified.

MC watchdog timer commands

The MC implements a standardized watchdog timer that can be used for a number of system timeout functions by system management software or by the BIOS. Setting a timeout value of 0 allows the selected timeout action to occur immediately. This provides a standardized means for devices on the IPMB, such as remote management cards, to perform emergency recovery actions.

Watchdog timer actions

The following actions are available on expiration of the watchdog timer:

- System reset
- System power off

- System power cycle
- Pre-timeout interrupt (optional)

The system reset on timeout, system power off on timeout, and system power cycle on timeout action selections are mutually exclusive. The watchdog timer is stopped whenever the system is powered-down. A command must be sent to start the timer after the system powers up.

Watchdog timer use field and expiration flags

The watchdog timer provides a timer use field that indicates the current use assigned to the watchdog timer. The watchdog timer provides a corresponding set of timer use expiration flags that are used to track the type of timeout that had occurred.

The timeout use expiration flags retain their state across system resets and power cycles, as long as the MC remains powered. The flags are normally cleared solely by the `set watchdog timer` command; with the exception of the don't log flag, which is cleared after every system hard reset or timer timeout.

The timer use fields indicate:

Timer use field	Description
BIOS FRB2 timeout	An FRB-2 (fault-resilient booting, level 2) timeout has occurred indicating that the last system reset or power cycle was due to the system timeout during POST, presumed to be caused by a failure or hang related to the bootstrap processor. ¹
BIOS POST timeout	In this mode, the timeout occurred while the watchdog timer was being used by the BIOS for some purpose other than FRB-2 or OS load watchdog.
OS load timeout	The last reset or power cycle was caused by the timer being used to watchdog the interval from boot to OS up and running. This mode requires system management software, or OS support. BIOS should clear this flag if it starts this timer during POST.
SMS OS watchdog timeout	Indicates that the timer was being used by SMS. During run-time, SMS starts the timer, then periodically resets it to keep it from expiring. This periodic action serves as a heartbeat that indicates that the OS (or at least the SMS task) is still functioning. If SMS hangs, the timer expires and the MC generates a system reset. When SMS enables the timer, it should make sure the SMS bit is set to indicate that the timer is being used in its OS watchdog role.
OEM	Indicates that the timer was being used for an OEM-specific function.

¹ In a multiprocessor system, the bootstrap processor is defined as the processor that, on system power-up or hard reset, is allowed to run and execute system initialization (BIOS POST) while the remaining processors are held in an idle state awaiting startup by the multiprocessing OS.

Using the timer use field and expiration flags

The software that sets the timer use field is responsible for managing the associated timer use expiration flag. For example, if SMS sets the timer use to SMS/OS watchdog, then that same SMS is responsible for acting on and clearing the associated timer use expiration flag.

In addition, software should only interpret or manage the expiration flags for watchdog timer uses that it set. For example, BIOS should not report watchdog timer expirations or clear the expiration flags for non-BIOS uses of the timer. This is to allow the software that did set the timer use to see that a matching expiration occurred.

Watchdog timer event logging

By default, the MC automatically logs the corresponding sensor-specific watchdog sensor event when a timer expiration occurs. A don't log bit is provided to temporarily disable the automatic logging. The don't log bit is automatically cleared (logging re-enabled) whenever a timer expiration occurs.

Pre-timeout interrupt

The watchdog timer offers a pre-timeout interrupt option. This option is enabled whenever the interrupt on timeout option is selected along with any of the other watchdog timer actions. If this option is enabled, the MC generates the selected interrupt a fixed interval before the timer expires. This feature can be used to allow an interrupt handler to intercept the timeout event before it actually occurs. The default pre-timeout interrupt interval is one (1) second.

The watchdog timeout action and the pre-timeout interrupt functions are individually enabled. Thus, the watchdog timer can be configured so that when it times out it provides just an interrupt, just the selected action, both an interrupt and selected action, or none.

If the pre-timeout interval is set to zero, the pre-timeout action occurs concurrently with the timeout action. If a power or reset action is selected with a pre-timeout interval of zero, there is no guarantee that a pre-timeout interrupt handler would have time to execute, or to run to completion.

Pre-timeout interrupt support detection

An application that wishes to use a particular pre-timeout interrupt can check for its support by issuing a `set watchdog timer` command with the desired pre-timeout interrupt selection. If the controller does not return an error completion code, then a `get watchdog timer` command should be issued to verify that the interrupt selection was accepted.

While it can be assumed that a controller that accepts a given interrupt selection supports the associated interrupt, it is recommended that, if possible, an application also generate a test interrupt and verify that the interrupt occurs and the handler executes correctly.

BIOS support for watchdog timer

If a system warm reset occurs, the watchdog timer may still be running while BIOS executes POST. Therefore, BIOS should take steps to stop or restart the watchdog timer early in POST. Otherwise, the timer may expire later during POST or after the OS has booted.

Reset watchdog timer command

This command is used for starting and restarting the watchdog timer from the initial countdown value that was specified in the `set watchdog timer` command. If a pre-timeout interrupt has been configured, the `reset watchdog timer` command is not restart the timer once the pre-timeout interrupt interval has been reached. The only way to stop the timer once it has reached this point is via the `set watchdog timer` command.

Table 60 Reset watchdog timer command response data

Response data byte number	Data field
1	Completion code. Generic plus command-specific completion code: 80h — Attempt to start un-initialized watchdog. It is recommended that a MC implementation return this error completion code to indicate to software that a <code>set watchdog timer</code> command has not been issued to initialize the timer since the last system power on, reset, or MC reset. Since many systems may initialize the watchdog timer during BIOS operation, this condition may only be seen by software if a MC gets re-initialized during system operation (as might be the case if a firmware update occurred, for example).

Set watchdog timer command

This command is used for initializing and configuring the watchdog timer. The command is also used for stopping the timer.

If the timer is already running, the `set watchdog timer` command stops the timer (unless the don't stop bit is set) and clears the watchdog pre-timeout interrupt flag. MC hard resets, system hard resets, and the `cold reset` command also stop the timer and clear the flag.

- Byte 1 is used for selecting the timer use and configuring whether an event will be logged on expiration.
- Byte 2 is used for selecting the timeout action and pre-timeout interrupt type.
- Byte 3 sets the pre-timeout interval. If the interval is set to zero, the pre-timeout action occurs concurrently with the timeout action.
- Byte 4 is used for clearing the timer use expiration flags. A bit set in byte 4 of this command clears the corresponding bit in byte 5 of the `get watchdog timer` command.
- Bytes 5 and 6 hold the least significant and most significant bytes, respectively, of the countdown value. The watchdog timer decrement is one count/100 ms. The counter expires when the count reaches zero. If the counter is loaded with zero and the `reset watchdog` command is issued to start the timer, the associated timer events occur immediately.

Table 61 Set watchdog timer command request and response data

Request data byte number	Data field		
1	Timer use		
	[7]	1b =	Don't log
	[6]	1b =	Don't stop the timer on <code>set watchdog timer</code> command. New parameters take effect immediately. If the timer is already running, the countdown value gets set to the given value and the countdown continues from that point. If the timer is already stopped, it remains stopped. If the pre-timeout interrupt bit is set, it is cleared. ¹
		0b =	Timer stops automatically when the <code>set watchdog timer</code> command is received.
	[5:3]	Reserved	
	[2:0]	Timer use (logged on expiration when don't log bit = 0b)	
		000b =	Reserved
		001b =	BIOS FRB2
		010b =	BIOS/POST
		011b =	OS Load
		100b =	SMS/OS
		101b =	OEM
		110b - 111b =	Reserved
2	Timer actions		
	[7]	Reserved	
	[6:4]	Pre-timeout interrupt (logged on expiration when don't log bit = 0b)	
		000b =	None
		001b =	SMI (optional)
		010b =	NMI / diagnostic interrupt (optional)
		011b =	Messaging interrupt (this is the same interrupt as allocated to the messaging interface, if communications interrupts are supported for the system interface).

Table 61 Set watchdog timer command request and response data *(continued)*

		100b,111b =	Reserved
	[3]		Reserved
	[2:0]		Timeout action
		000b =	No action
		001b =	Hard reset
		010b =	Power down
		011b =	Power dycle
		100b,111b =	Reserved
3			Pre-timeout interval in seconds. 1 based.
4			Timer use expiration flags clear. (0b = leave alone, 1b = clear timer use expiration bit).
	[7]		Reserved
	[6]		Reserved
	[5]		OEM
	[4]		SMS/OS
	[3]		OS load
	[2]		BIOS/POST
	[1]		BIOS FRB2
	[0]		Reserved
5			Initial countdown value, LS byte (100 ms/count)
6			Initial countdown value, MS byte
Response data byte number	Data field		
1			Completion code

¹ Potential race conditions exist with implementations of this option. If the `set watchdog timer` command is sent just before a pre-timeout interrupt or timeout is set to occur, the timeout could occur before the command is executed. To avoid this condition, it is recommended that software set this value no closer than 3 counts before the pre-timeout or timeout value is reached.

Get watchdog timer command

This command retrieves the current settings and present countdown of the watchdog timer. The timer use expiration flags in byte 5 retain their states across system resets and system power cycles. With the exception of bit 6 in the timer use byte, the timer use expiration flags are cleared using the `set watchdog timer` command. They may also become cleared because of a loss of MC power, firmware update, or other cause of MC hard reset. Bit 6 of the timer use byte is automatically cleared to 0b whenever the timer times out, is stopped when the system is powered down, enters a sleep state, or is reset.

Table 62 Get watchdog timer command response data

Response data byte number	Data field
1	Completion code

2	Timer use	
	[7]	1b = Don't log
	[6]	1b = Timer is started (running)
		0b = Timer is stopped
	[5:3]	Reserved
	[2:0]	Timer use (logged on expiration when don't log bit = 0b)
		000b = Reserved
		001b = BIOS FRB2
		010b = BIOS/POST
		011b = OS Load
		100b = SMS/OS
		101b = OEM
		110b-111b = Reserved
3	Timer actions	
	[7]	Reserved
	[6:4]	Pre-timeout interrupt (logged on expiration when don't log bit = 0b)
		000b = None
		001b = SMI (if implemented)
		010b = NMI / diagnostic interrupt (if implemented)
		011b = Messaging interrupt (this is the same interrupt as allocated to the messaging interface).
		100b,111b = Reserved
	[3]	Reserved
	[2:0]	Timeout action
		000b = No action
		001b = Hard reset
		010b = Power down
		011b = Power dycle
		100b,111b = Reserved
4	Pre-timeout interval in seconds. 1 based.	
5	Timer use expiration flags clear. (1b = timer expired while associated use was selected.)	
	[7]	Reserved
	[6]	Reserved
	[5]	OEM
	[4]	SMS/OS
	[3]	OS load

Table 62 Get watchdog timer command response data *(continued)*

	[2]	BIOS/POST
	[1]	BIOS FRB2
	[0]	Reserved
6	Initial countdown value, LS byte (100 ms/count)	
7	Initial countdown value, MS byte	
8	Present countdown value, LS byte. The initial countdown value and present countdown values should match immediately after the countdown is initialized via a <code>set watchdog timer</code> command and after a <code>reset watchdog timer</code> has been executed. Internal delays in the MC may require software to delay up to 100 ms before seeing the countdown value change and be reflected in the <code>get watchdog timer</code> command.	
9	Present countdown value, MS byte	

Chassis commands

The following chassis commands are specified for IPMI v1.5. These commands are primarily to provide standardized chassis status and control functions for remote management cards and remote consoles that access the MC. They can also be used for emergency management control functions by system management software.

Get chassis capabilities command

This command returns information about which main chassis management functions are present on the IPMB (or virtual IPMB) and what addresses are used to access those functions. This command is used to find the devices that provide functions such as SEL, SDR, and ICMB bridging so that they can be accessed via commands delivered via a physical or logical IPMB. The command does not include a channel number for the individual functions, therefore all reported functions must be located on the primary IPMB.

The chassis capabilities information is non-volatile. There is no requirement that the information be configurable. The chassis device function in a peripheral chassis may be hardcoded with this information. For example, a system that implements the ICMB as an add-on bridge to an MC is typically able to have the well known address for the MC (20h) hardcoded as the address for the chassis SDR, SEL, and SM devices, while the chassis FRU info device address could be set with the chassis devices own address.

An add-in device that serves as a bridge device that could be used in different vendors systems may want to provide a way for this information to be configured. The `set chassis capabilities` command is one option for providing this.

Table 63 Get chassis capabilities command response data

Response data byte number	Data field		
1	Completion code		
2	Capabilities flags		
	[7:4]	Reserved	
	[3]	1b =	Provides power interlock (IPMI 1.5)
	[2]	1b =	Provides diagnostic interrupt, FP NMI. (IPMI 1.5)
	[1]	1b =	Provides front panel lockout which indicates that the chassis has capabilities to lock out external power control and reset button or front panel interfaces and/or detect tampering with those interfaces.
	[0]	1b =	Chassis provides intrusion (physical security) sensor

Table 63 Get chassis capabilities command response data *(continued)*

3	Chassis FRU info device address. All IPMB addresses used in this command have the 7-bit I ² C slave address as the most-significant 7-bits and the least significant bit set to 0b. 00h = unspecified.
4	Chassis SDR device address
5	Chassis SEL device address
6	Chassis system management device address
(7)	Chassis bridge device address. Reports location of the ICMB bridge function. If this field is not provided, the address is assumed to be the MC address (20h). Implementing this field is required when the <code>get chassis capabilities</code> command is implemented by an MC, and whenever the chassis bridge function is implemented at an address other than 20h.

Get chassis status command

This command is available to ChMC and MC.

This command returns information regarding the high-level status of the system chassis and main power subsystem.

Table 64 Get chassis status command response data

Response data byte number	Data field	
1	Completion code	
2	Current power state	
	[7]	Reserved
	[6:5]	Power restore policy ¹
		00b = Chassis stays powered off after AC/mains returns
		01b = After AC returns, power is restored to the state that was in effect when AC/mains was lost
		10b = Chassis always powers up after AC/mains returns
		11b = Unknown
	[4]	Power control fault
		1b = Controller attempted to turn system power on or off, but system did not enter desired state.
	[3]	Power fault
		1b = Fault detected in main power subsystem.
	[2]	1b = Interlock (chassis is presently shut down because a chassis panel interlock switch is active). (IPMI 1.5).
	[1]	Power overload
		1b = System shutdown because of power overload condition
	[0]	Power is on
		1b = System power is on
		0b = System power is off (soft-off S4/S5 or mechanical off)
3	Last power event	
	[7:5]	Reserved
	[4]	1b = Last Power is on state was entered via IPMI command

Table 64 Get chassis status command response data *(continued)*

	[3]	1b =	Last power down caused by power fault
	[2]	1b =	Last power down caused by a power interlock being activated
	[1]	1b =	Last power down caused by a power overload
	[0]	1b =	AC failed
4	Miscellaneous chassis state		
	[7]	Reserved	
	[6]	1b =	Chassis identify command and state information supported (optional)
		0b =	Chassis identify command support unspecified via this command. (The <code>get command support</code> command, if implemented, would still indicate support for the <code>chassis identify</code> command).
	[5:4]	Chassis identify state. Mandatory when bit [6] = 1b, reserved (return as 00b) otherwise. Returns the present chassis identify state. See “ Chassis identify command ” (page 105).	
	[3]	1b =	Cooling/fan fault detected
	[2]	1b =	Drive fault
	[1]	1b =	Front panel lockout active (power off and reset via chassis push-buttons disabled.)
	[0]	1b =	Chassis intrusion active
(5)	Front panel button capabilities and disable/enable status (optional). Button actually refers to the ability for the local user to be able to perform the specified functions via a pushbutton, switch, or other front panel control built into the system chassis.		
	[7]	1b =	Standby (sleep) button disable allowed
	[6]	1b =	Diagnostic Interrupt button disable allowed
	[5]	1b =	Reset button disable allowed
	[4]	1b =	Power off button disable allowed (in the case there is a single combined power/standby (sleep) button, disabling power off also disables sleep requests via that button).
	[3]	1b =	Standby (sleep) button disabled
	[2]	1b =	Diagnostic Interrupt button disabled
	[1]	1b =	Reset button disabled
	[0]	1b =	Power off button disabled (in the case there is a single combined power/standby (sleep) button, then this indicates that sleep requests via that button are also disabled).

¹ In some installations, the chassis' main power feed may be DC based. For example, -48V. In this case, the power restore policy for AC/mains refers to the loss and restoration of the DC main power feed.

Chassis control command

This command is available to the MC.

This command provides a mechanism for providing power up, power down, and reset control.

Table 65 Chassis control command request and response data

Request data byte number	Data field	
1	[7:4]	Reserved
	[3:0]	Chassis control ¹

Table 65 Chassis control command request and response data *(continued)*

	0h =	Power down. Force system into soft off (S4/S45) state. This is for emergency management power down actions. The command does not initiate a clean shut-down of the operating system before powering down the system.
	1h =	Power up.
	2h =	Power cycle (optional). This command provides a power off interval of at least 1 second following the de-assertion of the system's power good status from the main power subsystem. It is recommended that no action occur if system power is off (S4/S5) when this action is selected, and that a D5h Request parameter(s) not supported in present state. error completion code be returned. Some implementations may cause a system power up if a power cycle operation is selected when system power is down. For consistency of operation, it is recommended that SMS first check the system power state before issuing a power cycle, and only issue the command if the system power is on or in a lower sleep state than S4/S5.
	3h =	Hard reset. In some implementations, the MC may not know whether a reset causes any particular effect and pulses the system reset signal regardless of power state. If the implementation can tell that no action occurs if a reset is delivered in a given power state, then it is recommended (but still optional) that a D5h Request parameter(s) not supported in present state. error completion code be returned. NOTE: Reset is not supported in Moonshot.
	4h =	Pulse diagnostic interrupt (optional). Pulse a version of a diagnostic interrupt that goes directly to the processor(s). This is typically used to cause the operating system to do a diagnostic dump (OS dependent). The interrupt is commonly an NMI on IA-32 systems and an INIT on Intel® Itanium™ processor based systems.
	5h =	Initiate a soft-shutdown of OS via ACPI by emulating a fatal overtemperature (optional).
	All other =	Reserved
Response data byte number	Data field	
1	Completion code ²	

¹ The command can also be used for compute blades or compute partition applications where the blades or partitions entities are emulating independent computer systems that implement IPMI. In these applications, the chassis power control aspects of the command are not required to be supported. Individual blades or computer partitions can elect to either not support the power on/off functions, can use them for power control of the blade/partition independent of the containing chassis, or may map them into a power control scheme for the overall chassis. For example, a scheme where chassis power will go off only after all blades within a chassis have been commanded into the power off state.

² The implementation is allowed to return the completion code before performing the selected control action if necessary.

Chassis identify command

This command is available to the MC.

This command causes the chassis to physically identify itself by a mechanism chosen by the system implementation; such as turning on blinking user-visible lights or emitting beeps via a speaker, LCD panel, etc. Unless the optional force identify on capability is supported and used, the `chassis identify` command automatically times out and deasserts the indication after a configurable time-out. Software must periodically resend the command to keep the identify condition asserted. This restarts the timeout.

NOTE: In Moonshot, directing this command at the chassis affects the chassis UID, while directing the command at a cartridge affects the cartridge UID.

Table 66 Chassis identify command request and response data

Request data byte number	Data field		
1 ¹	[7:0]	Identify interval in seconds (optional). 1-based. Timing accuracy = -0/+20%. If this byte is not provided, the default timeout shall be 15 seconds -0/+20%. This byte can be overridden by optional byte 2. 00h = Turn off identify	
2 ²	Force identify on (optional). This field enables software to command the identify to be on indefinitely. The MC implementation should return an error completion code if this byte is not supported.		
	[7:1]	Reserved	
	[0]	1b =	Turn on identify indefinitely. This overrides the values in byte 1.
		0b =	Identify state driven according to byte 1.
Response data byte number	Data field		
1	Completion code		

¹ This parameter byte is optionally present. If not provided, the chassis identify can be used to turn on the identify indication for the default timeout interval, but cannot be used to turn the indication off.

² This parameter byte is optionally present. If provided, it is highly recommended that the chassis provides a local manual mechanism that enables a user or service personnel to turn off Identify. If a local manual mechanism is not provided, AC removal (MC reset) should remove the indication.

Set power restore policy command

This command is available to the MC.

This command can be used to configure the power restore policy. This configuration parameter is kept in non-volatile storage. The power restore policy determines how the system or chassis behaves when AC power returns after an AC power loss. The `get chassis status` command returns the power restore policy setting.

Table 67 Set power restore policy command request and response data

Request data byte number	Data field		
1	[7:3]	Reserved	
	[2:0]	Power restore policy ¹	
		011b =	No change (just get present policy support)
		010b =	Chassis always powers up after AC/mains is applied or returns
		001b =	After AC/mains is applied or returns, power is restored to the state that was in effect when AC/mains was removed or lost
		000b =	Chassis always stays powered off after AC/mains is applied, power push button or command required to power on system
		All other =	Reserved
Response data byte number	Data field		
1	Completion code. A non-zero completion code should be returned if an attempt is made to set a policy option that is not supported.		
2	Power restore policy support (bitfield)		
	[7:3]	Reserved	

Table 67 Set power restore policy command request and response data *(continued)*

	[2]	1b =	Chassis supports always powering up after AC/mains returns
	[1]	1b =	Chassis supports restoring power to the state that was in effect when AC/mains was lost
	[0]	1b =	Chassis supports staying powered off after AC/mains returns

¹ In some installations, the chassis' main power feed may be DC based. For example, -48V. In this case, the power restore policy for AC/mains refers to the loss and restoration of the DC main power feed.

Set system boot options command

This command is available to the MC.

This command is used to set parameters that direct the system boot following a system power up or reset. The boot flags only apply for one system restart. It is the responsibility of the system BIOS to read these settings from the MC and then clear the boot flags.

It is possible that a remote console application could set the boot option flags and then be terminated either accidentally or intentionally. In this circumstance, it is possible that a user-initiated system restart could occur hours or even days later. If the boot options were used without examining the reset cause, this could cause an unexpected boot sequence. Thus, the MC automatically clears a boot flags valid bit if a system restart is not initiated by a `chassis control` command within 60 seconds +/- 10% of the valid flag being set. The MC also clears the bit on any system resets or power cycles that are not triggered by a `system control` command. This default behavior can be temporarily overridden using the MC boot flag valid bit clearing parameter.

Table 68 Set system boot options command request and response data

Request data byte number	Data field		
1	Parameter valid		
	[7]	1 b =	Mark parameter invalid/locked
		0 b =	Mark parameter valid/unlocked
	[6:0]	Boot option parameter selector	
(2:N)	Boot option parameter data. Passing 0-bytes of parameter data allows the parameter valid bit to be changed without affecting the present parameter setting. NOTE: Moonshot currently supports only parameters 0, 4, and 5. See Table 70 (page 108) .		
Response data byte number	Data field		
1	Completion code. Generic plus the command-specific completion codes:		
	80h =	Parameter not supported.	
	81h =	Attempt to set the set in progress value (in parameter #0) when not in the set complete state. This completion code provides a way to recognize that another party has already claimed the parameters.	
	82h =	Attempt to write read-only parameter.	

Get system boot options command

This command is available to the MC.

This command is used to retrieve the boot options set by the `set system boot options` command.

NOTE: Moonshot currently supports only parameters 0, 3, 4, and 5. See [Table 70 \(page 108\)](#).

Table 69 Get system boot options request command and response data

Request data byte number	Data field		
1	Parameter selector		
	[7]	Reserved	
	[6:0]	Boot option parameter selector	
2	[7:0] - Set selector. Selects a particular block or set of parameters under the given parameter selector. Write as 00h if parameter does not use a set selector.		
3	[7:0] - Block selector. Selects a particular block within a set of parameters. Write as 00h if parameter does not use a block selector. NOTE: There are no IPMI-specified boot options parameters that use the block selector. However, this field is provided for consistency with other configuration commands and as a placeholder for future extension of the IPMI specification.		
Response data byte number	Data field		
1	Completion code. Generic plus the command-specific completion code: 80h = parameter not supported.		
2	[7:4]	Reserved	
	[3:0]	Parameter version. 1h for this specification unless otherwise specified.	
3	Parameter valid		
	[7]	1b =	Parameter marked invalid / locked
		0b =	Parameter marked valid / unlocked
	[6:0]	Boot option parameter selector	
4:N	Configuration parameter data, per Table 70 (page 108) . If the rollback feature is implemented, the MC makes a copy of the existing parameters when the set in progress state becomes asserted (see the set in progress parameter #0). While the set in progress state is active, the MC returns data from this copy of the parameters, plus any uncommitted changes that were made to the data. Otherwise, the MC returns parameter data from non-volatile storage.		

Table 70 Boot option parameters

Parameter	#	Parameter data (non-volatile unless otherwise noted)		
Set in progress (volatile)	0	Data 1 - This parameter is used to indicate when any of the following parameters are being updated, and when the updates are completed. The bit is primarily provided to alert software that some other software or utility is in the process of making changes to the data. An implementation can also elect to provide a rollback feature that uses this information to decide whether to roll back to the previous configuration information, or to accept the configuration change. If used, the roll back restores all parameters to their previous state. Otherwise, the change takes effect when the write occurs.		
		[7:2]	Reserved	
		[1:0]	00b =	Set complete. If a system reset or transition to powered down state occurs while set in progress is active, the MC goes to the set complete state. If rollback is implemented, going directly to set complete without first doing a commit write causes any pending write data to be discarded.
			01b =	Set in progress. This flag indicates that some utility or other software is presently doing writes to

Table 70 Boot option parameters *(continued)*

Parameter	#	Parameter data (non-volatile unless otherwise noted)		
				parameter data. It is a notification flag only, it is not a resource lock. The MC does not provide any interlock mechanism that would prevent other software from writing parameter data.
			11b =	Reserved
MC boot flag valid bit clearing (semi-volatile) ¹	3	Data 1 — MC boot flag valid bit clearing. Default = 00000b.		
		[7:5]	Reserved	
		[4]	1b =	Do not clear valid bit on reset/power cycle caused by PEF. Not supported in Moonshot.
		[3]	1b =	Do not automatically clear boot flag valid bit if chassis control command is not received within 60-second timeout (countdown restarts when a chassis control command is received).
		[2]	1b =	Do not clear valid bit on reset/power cycle caused by watchdog timeout.
		[1]	1b =	Do not clear valid bit on push button reset/soft-reset (such as "Ctrl-Alt-Del"). Not supported in Moonshot.
		[0]	1b =	Do not clear valid bit on power up via power push button or wake event. Not supported in Moonshot.
Boot info acknowledge (semi-volatile) ¹	4	These flags are used to allow individual parties to track whether they have already seen and handled the boot information. Applications that deal with boot information should check the boot information and clear their corresponding bit after consuming the boot options data.		
		Data 1: Write mask (write-only). This field is returned as 00h when read. This is to eliminate the need for the MC to provide storage for the write mask field.		
		[7]	1b =	Enable write to bit 7 of data field
		[6]	1b =	Enable write to bit 6 of data field
		[5]	1b =	Enable write to bit 5 of data field
		[4]	1b =	Enable write to bit 4 of data field
		[3]	1b =	Enable write to bit 3 of data field
		[2]	1b =	Enable write to bit 2 of data field
		[1]	1b =	Enable write to bit 1 of data field
		[0]	1b =	Enable write to bit 0 of data field
		Data 2: Boot initiator acknowledge data. The boot initiator should typically write FFh to this parameter before initiating the boot. The boot initiator may write 0's if it wants to intentionally direct a given party to ignore the boot info. This field is automatically initialized to 00h when the management controller is first powered up or reset.		
		[7]	Reserved. Write as 1b. Ignore on read.	
		[6]	Reserved. Write as 1b. Ignore on read.	
		[5]	Reserved. Write as 1b. Ignore on read.	
		[4]	0b =	OEM has handled boot information.
		[3]	0b =	SMS has handled boot information.
		[2]	0b =	OS/service partition has handled boot information.

Table 70 Boot option parameters *(continued)*









Parameter	#	Parameter data (non-volatile unless otherwise noted)		
Boot flags (semi-volatile) ¹	5	[1]	0b =	OS loader has handled boot information.
		[0]	0b =	BIOS/POST has handled boot information.
		Data 1		
		[7]	1b =	Boot flags valid. The bit should be set to indicate that valid flag data is present. This bit may be automatically cleared based on the boot flag valid bit clearing parameter.
		[6]	0b =	Options apply to next boot only.
			1b =	Options requested to be persistent for all future boots (such as requests for BIOS to change its boot settings). NOTE: In order to set this bit remotely (over a session), the user must execute the <code>set system boot options</code> command at Admin privilege level. In order to retain backward compatibility, this bit is automatically cleared by the MC whenever the boot flags valid bit is clear (0b). This is to avoid the possibility that this bit would already be set when an older application changes other options. Thus, this bit and the boot flags valid bit must be set simultaneously.
		[5]	BIOS boot type (for BIOS that support both legacy and EFI boots).	
			0b =	PC compatible boot (legacy).
			1b =	Extensible firmware interface boot (EFI). Not supported in Moonshot.
		[4:0]	Reserved	
		BIOS support for the following flags is optional. If a given flag is supported, it must cause the specified function to occur in order for the implementation to be considered conformant with this specification. The following parameters represent temporary overrides of the BIOS default settings when data1[6] has value 0b (one-boot), and represent requests to persistently change the BIOS boot behavior when data1[6] has value 1b (persistent). BIOS should only use the following flags when the boot flags valid bit (data1[7]) is set (1b). If data[6] = 0b (one-boot) a value of 0 for a given data2 parameter indicates that BIOS should use its default configuration for the given option (no override) - a non-zero value requests BIOS to enter the requested state. If data[6] = 1b (persistent) BIOS is requested to change its setting according to the flag. This only applies to parameters labeled  . Settings for other parameters are ignored.		
		Data 2		
		[7]	1b =	CMOS clear. Not supported in Moonshot.
		[6]	1b =	Lock keyboard. Not supported in Moonshot.
		[5:2]	Boot device selector 	
			0000b =	No override
			0001b =	Force PXE

Table 70 Boot option parameters *(continued)*

Parameter	#	Parameter data (non-volatile unless otherwise noted)	
			0010b = Force boot from default hard-drive ²
			1100-1110b Reserved
		[1]	1b = Screen blank. Not supported in Moonshot.
		[0]	1b =  Lock out reset buttons. Not supported in Moonshot.
		Data 3	
		[7]	1b =  Lock out (power off/ sleep request) via power button. Not supported in Moonshot.
		[6:5]	Firmware (BIOS) verbosity (directs what appears on POST display). Not supported in Moonshot.
		[4]	 0b (1b = Force progress event traps for [IPMI 2.0]). Not supported in Moonshot.
		[3]	1b =  User password bypass . Not supported in Moonshot.
		[2]	1b =  Lock out sleep button . Not supported in Moonshot.
		[1:0]	 Console redirection control. Not supported in Moonshot.
OEM parameters (optional). Non-volatile or volatile as specified by OEM.	96:127	This range is available for special OEM configuration parameters. The OEM is identified according to the manufacturer ID field returned by the <code>get device ID</code> command.	

¹ Semi-volatile means that the parameter is kept across system power cycles, resets, system power on/off, and sleep state changes, but is not preserved if the management controller loses standby power or is cold reset. Parameters designated as semi-volatile are initialized to 0's upon controller power up or hard reset, unless otherwise specified.

² IPMI allows software to use the boot initiator mailbox as a way for a remote application to pass OEM parameters for additional selection of the boot process and direction of the startup of post-boot software. If additional parameters are not included, the system boots the primary/first-scanned device of the type specified.

Get POH counter command

This command is available to the MC.

IPMI provides a specification for an optional, POH counter. The management controller automatically increments non-volatile storage at the specified rate whenever the system is powered up. It is recommended that this command be implemented in the MC to provide a standardized location for this function.

The definition of powered up used in this document indicates that the power-on hours accumulate whenever the system is in the operational (S0) state. An implementation may elect to increment power-on hours in the S1 and S2 states as well.

Clear or set commands are not specified for this counter because the counter is most typically used for warranty tracking or replacement purposes.

Table 71 Get POH counter command response data

Response data byte number	Data field
1	Completion code
2	Minutes per count
3:6	Counter reading. LS byte first.

When the system is powered down between counts, the counter either picks up incrementing at the offset at which the power down occurred, or starts counting at 0 minutes from the last counter reading, depending on the choice of the implementer. In any case, the time does not get rounded up to the next count as a result of powering down between counts.

Event commands

The sensor/event network function is used for device functionality related to the transmission, reception, and handling of event messages and platform sensors. An event message is actually a sensor/event message with a command byte of '02h'. The request is also referred to as an event request message, while the corresponding response is referred to as an event response message.

Set event receiver command

This command tells a controller where to send event messages. The slave address and LUN of the event receiver must be provided. A value FFh for the event receiver slave address disables event message generation entirely. This command is only applicable to management controllers that act as IPMB event generators.

A device that receives a `set event receiver` command re-arms event generation for all its internal sensors. This means internally re-scanning for the event condition and updating the event status based on the result. This causes devices that have any pre-existing event conditions to transmit new event messages for those events.

A reading/state unavailable (formerly initial update in progress) bit is provided with the `get sensor reading` and `get sensor event status` commands to help software avoid getting incorrect event status due to a re-arm. For example, a controller only scans for an event condition once every four seconds. Software that accessed the event status using the `get sensor reading` command could see the wrong status for up to four seconds before the event status would be correctly updated. A controller that has slow updates must implement the reading/state unavailable bit, and should not generate event messages until the update has completed. Software should ignore the event status bits while the reading/state unavailable bit is set.

Table 72 Set event receiver command request and response data

Request data byte number	Data field
1	Event receiver slave address. 0FFh disables event message generation. Otherwise: <ul style="list-style-type: none"> [7:1] - IPMB (I²C) slave address [0] - always 0b when [7:1] holds I²C slave address
2	<ul style="list-style-type: none"> [7:2] - reserved [1:0] - event receiver LUN
Response data byte number	Data field
1	Completion code

Get event receiver command

This command is used to retrieve the present setting for the event receiver slave address and LUN. This command is only applicable to management controllers that act as IPMB event generators.

Table 73 Get event receiver command response data

Response data byte number	Data field
1	Completion code
2	Event receiver slave address. OFFh indicates event message generation has been disabled. Otherwise: <ul style="list-style-type: none">[7:1] - IPMB (I²C) slave address[0] - always 0b when [7:1] holds I²C slave address
3	<ul style="list-style-type: none">[7:2] - reserved[1:0] - event receiver LUN

Platform event message command

This command is available to ChMC and MC.

This command is a request for the MC to process the event data that the command contains. Typically, the data is logged to the SEL. Depending on the implementation, the data may also go to the event message buffer and processed by PEF.

In Moonshot, the Zone manager can receive event messages from the chassis, power supply and cartridge controllers. The System Node can receive event messages from the system interface as well as generate events for its own consumption.

Table 74 Platform event message command request and response data

IPMB messaging (IPMB, LAN, Serial/Modem, PCI Mgmt. Bus)		System interface	
Request data byte number	Data field	Request data byte number	Data field
—	Generator ID (RqSA, RqLUN)	1	Generator ID
1	EvMRev	2	EvMRev
2	Sensor type	3	Sensor type
3	Sensor #	4	Sensor #
4	Event Dir Event type	5	Event Dir Event type
5	Event data 1	6	Event data 1
6	Event data 2	7	Event data 2
7	Event data 3	8	Event data 3
Response data byte number		Response data byte number	
1	Completion code	1	Completion code

The generator ID field is a required element of an event request message. For IPMB messages, this field is equated to the requester's slave address and LUN fields. Thus, the generator ID information is not carried in the data field of an IPMB request message.

For system side interfaces, do not overlay the generator ID field with the message source address information. It should be specified as being carried in the data field of the request.

SEL commands

The SEL is a non-volatile repository for system events and certain system configuration information. The device that fields these commands is referred to as the SEL device. Event message information is normally written into the SEL after being received by the event receiver functionality in the event receiver device.

The SEL device commands are structured in such a way that the device can be separated from the event receiver device. In this instance, the event receiver device must send the appropriate `Add sel entry` message directly to the SEL device or pass the equivalent request through an intermediary.

SEL entries have a unique record id field, used for retrieving log entries from the SEL. SEL reading is done in a random access manner, that is, SEL entries are read in any order as long as the record id is known.

NOTE: SEL record id's 0000h and FFFFh are reserved for functional use and are not legal id values. Record ids are handles and are not required to be sequential or consecutive. Applications should not assume that the SEL record id will follow any particular numeric order.

SEL records are stored as ordered lists. Appending and deleting individual entries does not change the access order.

SEL device commands

The SEL device can be implemented as a separate device from the event receiver and event generator devices. If this is done, it is up to the implementer to create the method by which even messages are passed from the even receiver device to the SEL device.

Get SEL info command

This command return the number of entries in the SEL, SEL command version and the timestamp for the most recent entry and delete/clear. The most recent addition timestamp field returns the timestamp for the last add or log operation while the most recent erase field returns the timestamp for the last delete or clear operation.

These timestamps are independent of timestamps that may be returned by other commands, such as those returned by the `Get sdr repository info` command. The timestamp reflects when the most recent SEL add or erase occurred, and not when the last add or erase occurred on the physical storage device.

For example, the SEL Info most recent addition timestamp would reflect the last time a new event was added to the SEL. This would be independent of the Info most recent addition time for an SDR even if the implementation elected to implement the SEL and SDR repository in the same storage device.

Table 75 Get SEL info command request and response data

Request data byte number	Data field
1	Completion code <ul style="list-style-type: none">• 81h = cannot execute command• SEL erase in progress
2	SEL version — version number of the SEL command set for this SEL Device. <ul style="list-style-type: none">• 51h• BCD encoded with bits 7:4 holding the least significant digit of the revision and bits 3:0 holding the most significant bits.
3	Entries LS byte — number of log entries in SEL, LS byte

Table 75 Get SEL info command request and response data *(continued)*

Request data byte number	Data field
4	Entries MS byte — number of log entries in SEL, MS byte
5:6	Free space in bytes, LS byte first. FFFFh indicates 65535 or more bytes of free space are available.
7:10	Most recent addition timestamp. <ul style="list-style-type: none"> • LS byte first. • Returns FFFF_FFFFh if no SEL entries have ever been made or if a component update or error caused the retained value to be lost.
11:14	Most recent erase timestamp. Last time that one or more entries were deleted from the log. LS byte first.
15	Operation support <ul style="list-style-type: none"> • [7] — Overflow flag. 1=events have been dropped due to lack of space in the SEL. • [6:4] — reserved. Write as 000 • [3] — 1b = Delete SEL command supported • [2] — 1b = Partial Add SEL Entry command supported • [1] — 1b = Reserve SEL command supported • [0] — 1b = Get SEL Allocation information command supported

Reserve SEL command

This command sets the present owner of the SEL as identified by the software id or by the requesters slave address from the command. The reservation process provides a limited amount of protection on repository access from the IPMB when records are deleted or incrementally read.

The `reserve sel` command provides helps prevent the wrong record from being deleted. It includes a mechanism that prevents the SEL from being cleared when a new event is received in addition to preventing receipt of incorrect data during incremental reads.

The `reserve sel` does not guarantee access to the SEL. Essentially, this command prevents requesters from causing deadlocking.

A reservation id value is returned in response to this command. This value is required in other requests, such as the `clear sel` command. This commands will not execute unless the correct reservation id value is provided.

The reservation id is used in the following manner. Suppose an application wishes to clear the SEL. The application would first reserve the repository by issuing a `reserve sel` command. The application would then check that all SEL entries have been handled prior to issuing the `clear sel` command.

If a new event is placed in the SEL after records were checked, but before the `clear sel` command, it is possible for the event to be lost. However, the addition of a new event to the SEL causes the present reservation id to be cancelled. This would prevent the `clear sel` command from executing. If this occurred, the application would repeat the reserve check clear process until successful.

Table 76 Reserve SEL command request and respond data

Request data byte number	Data field
1	Completion code. 81h = cannot execute command, SEL erase in progress
2	Reservation id, LS byte 0000h reserved
3	Reservation id, MS byte

Get SEL entry command

Use this command to retrieve entries from the SEL. The record data field in the response returns the 16 bytes of data from the SEL event record.

Table 77 Get sel entry request data

Request data byte number	Data field
1:2	Reservation id, LS byte first. Only required for partial get. Use 0000h otherwise. ¹
3:4	SEL record id, LS byte first. 0000h = get first entry FFFFh — get last entry
5	Offset into record
6	Bytes to read. FFh means read entire record.

¹ The reservation id should be set to 0000h for implementations that don't implement the `reserve sel` command.

Table 78 Get set entry response data

Response data byte number	Data field
1	Completion code. Return an error completion code if the SEL is empty. 81h = cannot execute command, SEL erase in progress.
2:3	Next SEL record id, LS byte first (return FFFFh if the record returned is the last record.) FFFFh is not a valid record id.
4:N	Record data, 16 bytes for entire record.

Add SEL entry command

This command enables the BIOS to add records to the system event log. Normally, the SEL device and the event receiver device are incorporated into the same management controller. In this case, the BIOS or the system SMI handler adds its own events to the SEL by formatting an event message and transmitting it to the SEL device instead of using this command.

Records are added after the last record in the SEL. The SEL device adds the timestamp according to the SEL record type when it creates the record. In some cases, the timestamp bytes in the record data are ignored, there are still dummy timestamp bytes present in the data.

The record data field is passed in the request consists of all bytes of the SEL event record. The record id field that is passed in the request is just a placeholder. The record id field that was passed in the request is overwritten with a record id value that the SEL device generates before the record is stored. Depending on the record type, the entry may also be automatically timestamped. If the entry is automatically timestamped, the SEL device also overwrites the four bytes of the records timestamp field.

NOTE: The normal mechanism for adding entries to the SEL is by an event request message to the event receiver device.

Table 79 Add SEL entry request data

Request data byte number	Data field
1:16	Record data, 16 bytes

Table 80 Add SEL entry response data

Request data byte number	Data field
1	Completion code. Generic, plus following command specific: 80h = operation not supported for this record type 81h = cannot execute command, SEL erase in progress
2:3	Record id for added record, LS byte first.

Clear SEL

This command erases all contents of the System Event Log. Since this process may take several seconds, based on the type of storage device, the command also provides a means for obtaining the status of the erasure.

Table 81 Clear SEL entry request data

Request data byte number	Data field
1:2	Reservation ID, LS Byte first. ¹
3	'C' (43h)
4	'L' (4Ch)
5	'R' (52h)
6	AAh = initiate erase. 00h = get erasure status.

¹ The reservation ID should be set to 0000h for implementations that don't implement the Reserve SEL command.

Table 82 Clear SEL entry response data

Request data byte number	Data field
1	Completion Code
2	Erasure progress. [7:4] - reserved [3:0] - erasure progress <ul style="list-style-type: none"> 0h = erasure in progress. 1h = erase completed.

SEL record type ranges

Table 83 SEL record type ranges

Record ranges	Description
00h — BFh	This range is reserved for standard SEL record types. Records are automatically timestamped unless otherwise indicated.
C0h — DFh	This range is reserved for timestamped OEM SEL records. These records are automatically timestamped by the SEL device.
E0h — FFh	This range is reserved for non-timestamped OEM SEL records. The SEL device does not automatically timestamp these records. The four bytes passed in the byte locations for the timestamp are directly entered into the SEL.

Get SEL time command

This command returns the time from the SEL device. This time is used by the SEL device for event timestamping.

Table 84 Get SEL time command request and respond data

Response data byte number	Data field
1	Completion code
2:5	Present timestamp clock reading. LS byte first.
Request data byte number	Data field
1:4	Time in four byte format. LS byte first.

Set SEL time command

This command initializes the time in the SEL device. This time is used by the SEL device for event timestamping.

Table 85 Set SEL time command response data

Response data byte number	Data field
1	Completion code.

SDR repository device commands

The following sections describe the commands that an SDR repository device provides for accessing the SDR repository. The commands are designed to simplify the SDR repository device's implementation by pushing back intelligence to higher-level software where possible. The SDR repository device is not intended to be a database engine. Thus, the SDR access commands do not include automatic search functions. It is recommended that an application read the SDR repository into a RAM buffer and work from that copy (keeping track of the SDR timestamp to check for possible changes to the SDR repository). The general procedure for reading SDRs from the SDR repository is described under the `get sdr` command.

As with event messages, the commands are designed so that the SDR repository device is isolated and does not need to know the content and format of the SDR records themselves.

SDR record IDs

In order to generalize SDR access, sensor data records are accessed using a record ID number. There are a fixed number of possible record IDs for a given implementation of the SDR repository. The most common implementation of record IDs is as a value that translates directly to an index or offset into the SDR repository. However, it is also possible for an implementation to provide a level of indirection, and implement record IDs as handles to the SDRs.

Record ID values may be recycled so that the record ID of a previously deleted SDR can be used as the record ID for a new SDR. The requirement is that, at any given time, the record IDs are unique for all SDRs in the repository.

Record IDs can be reassigned by the SDR repository device as needed when records are added or deleted. An application that uses a record ID to directly access a record should always verify that the retrieved record information matches up with the ID information (slave address, LUN, sensor ID, and so on) of the desired sensor. An application that finds that the SDR at a given record ID has moved needs to re-enumerate the SDRs by listing them out using a series of `get sdr` commands. It is not necessary to read out the full record data to see if the record ID for a particular record has

changed. Software can determine whether a given record has been given a different record ID by examining just the SDR's header and record key bytes.

Get SDR repository info command

This command is available to the MC.

At the zone level, remember to issue the `SDR repository` version of the command. At any other zone, use the `device SDR` version of the command.

This command returns the SDR command version for the SDR repository. It also returns a timestamp for when the last add, delete, or clear occurred. The most recent addition timestamp field returns the timestamp for the last addition operation, while the most recent erase field returns the timestamp for the last delete or clear operation.

These timestamps are independent of timestamps that may be returned by other commands, such as those returned by the `get SEL info` command. The timestamp reflects when the most recent SDR repository add or erase occurred, not when the last add or erase occurred on the physical storage device.

For example, the SDR repository info most recent addition timestamp would reflect the last time a new record was added to the SDR repository. The SDR repository's most recent addition timestamp is always independent of the most recent addition time for the SEL - even if the SEL and SDR repository are implemented in the same physical storage device.

Table 86 Get SDR repository info command response data

Response data byte number	Data field		
1	Completion code		
2	SDR version - version number of the SDR command set for the SDR device. 51h for this specification. (BCD encoded with bits 7:4 holding the least significant digit of the revision and bits 3:0 holding the most significant bits.)		
3	Record count LS byte - number of records in the SDR repository.		
4	Record count MS Byte - number of records in the SDR repository.		
5:6	Free space in bytes, LS Byte first. 0000h indicates full, FFFEh indicates 64KB-2 or more available. FFFFh indicates unspecified.		
7:10	Most recent addition timestamp. LS byte first.		
11:14	Most recent erase (delete or clear) timestamp. LS byte first.		
15	Operation support		
	[7]	Overflow flag. 1=SDR could not be written due to lack of space in the SDR repository.	
	[6:5]	00b =	Modal/non-modal SDR repository update operation unspecified.
		01b =	Non-modal SDR repository update operation supported.
		10b =	Modal SDR repository update operation supported.
		11b =	Both modal and non-modal SDR repository update supported.
	[4]	Reserved. Write as 0b.	
	[3]	1b=	Delete SDR command supported.
	[2]	1b=	Partial Add SDR command supported.
	[1]	1b=	Reserve SDR repository command supported.
	[0]	1b=	Get SDR repository allocation information command supported.

Get SDR repository allocation info command

This command is available to the MC.

This command returns the number of possible allocation units, the amount of usable free space (in allocation units), the allocation unit size (in bytes), and the size of the largest contiguous free region (in allocation units). The allocation unit size is the number of bytes in which storage is allocated. For example, if a 20-byte record is to be added, and the SDR repository has a 16-byte allocation unit size, then the record would take up 32-bytes of storage.

The SDR repository implementation, at a minimum, provides an allocation unit size of 16 bytes or more and a maximum record size supporting a record of 64 bytes or more. Software should assume an allocation unit size of 16 bytes if this command is not implemented.

Table 87 Get SDR repository allocation info command response data

Response data byte number	Data field
1	Completion code
2	Number of possible allocation units, LS byte.
3	Number of possible allocation units, MS byte. This number indicates whether the total number of possible allocation units is equal to or less than the log size divided by the allocation unit size. 0000h indicates unspecified.
4	Allocation unit size in bytes. 0000h indicates unspecified.
5	
6	Number of free allocation units, LS byte.
7	Number of free allocation units, MS byte.
8	Largest free block in allocation units, LS byte.
9	Largest free block in allocation units, MS byte.
10	Maximum record size in allocation units.

Reserve SDR repository command

This command is available to the MC.

This command is used to set the present owner of the repository, as identified by the software ID or by the requester's slave address from the command. The reservation process provides a limited amount of protection on repository access from the IPMB when records are being deleted or incrementally read.

The `reserve SDR repository` command is provided to help prevent deleting the wrong record when doing deletes, and to prevent receiving incorrect data when doing incremental reads. It does not guarantee access to the SDR repository so that a pair of requesters could vie for access to the SDR that they alternately cancel the reservation that is held by the other - effectively deadlocking each other.

A reservation ID value is returned in response to this command which is required in other requests, such as the `delete SDR` command. These commands do not execute unless the correct reservation ID value is provided.

The reservation ID is used in the following manner. Suppose an application wishes to delete a particular record. The application would first reserve the repository by issuing a `reserve SDR repository` command. The application reads the header and key information from the record to verify that it has the correct record ID for the record. Assuming this is correct, the application issues a `delete SDR` command using the reservation ID and record ID as parameters.

If an event had occurred that changed the record IDs after the header and key information was read but before the `delete SDR` command, the `delete SDR` command could be issued with

the record ID for the wrong record. However, events that change record IDs for any existing records cause the present reservation ID to be canceled. This prevents software from using an out-of-date record ID to access a record. For example, it would prevent the `delete SDR` command from executing and deleting the wrong record in case a given record ID was reassigned to a different record.

Table 88 Reserve SDR repository command response data

Response data byte number	Data field
1	Completion code
2	Reservation ID, LS byte.
3	Reservation ID, MS byte.

Reservation restricted commands

A requester must issue a `reserve SDR repository` command before issuing any of the following SDR repository commands. This command only needs to be reissued if the reservation is canceled. The following commands are rejected if the requester's reservation has been canceled.

- Delete SDR command
- Clear SDR Repository command
- Get SDR command (if a partial read)

If the given reservation has been canceled, a reservation canceled completion code is returned in response to the above commands. See [“Reservation cancellation” \(page 121\)](#)

Since record IDs could change between offset 0 “gets” of a given record, it is the responsibility of the device accessing the repository to verify that the retrieved record information matches up with the ID information (slave address, LUN, sensor ID, and so on) of the desired sensor.

Reservation cancellation

The SDR repository device automatically cancels the present SDR repository reservation after any of the following events occur:

- An SDR record is added using the `add SDR` command so that other record IDs change. As a simplification, an implementation is allowed to cancel the reservation on any SDR record add.
- An SDR record is deleted so that other record IDs change. As a simplification, an implementation is allowed to cancel the reservation on any SDR record deletion.
- The SDR repository is cleared.
- The SDR repository device is reset (via hardware or `cold reset` command).
- A new `reserve SDR repository` command is received.

An error completion code is returned if an attempt is made to execute a command that requires a reservation ID, but the reservation ID used is not valid or current.

Get SDR command

This command is available to the MC.

This command returns the sensor record specified by record ID. The command also accepts a byte range specification that allows just a selected portion of the record to be retrieved (incremental read). The requester must first reserve the SDR repository using the `reserve SDR repository` command in order for an incremental read to an offset other than 0000h to be accepted. (It is also recommended that an application use the `get SDR repository info` command to verify the

version of the SDR repository before it sends any other SDR repository commands. This is important since the SDR repository command format and operation can change between versions).

If the record ID is specified as 0000h, this command returns the record header for the first SDR in the repository. FFFFh specifies that the last SDR in the repository should be listed. If the record ID is non-zero, the command returns the information from the matching record, and the record ID for the next SDR in the repository.

An application that wishes to retrieve the full set of SDR records must first issue the `get SDR` command starting with 0000h as the record ID to get the first record. The next record ID is extracted from the response and this is then used as the record ID in a `get SDR` request to get the next record. This is repeated until the last record ID value (FFFFh) is returned in the next record ID field of the response.

A partial read from offset 0000h into the record can be used to extract the header and associated key fields for the specified sensor data record in the SDR repository. An application can use the command in this manner to get a list of what records are in the SDR and to identify the instances of each type. It can also be used to search for a particular sensor record.

NOTE: To support future extensions, applications should check the SDR Vversion byte before interpreting any of the data that follows.

The application issuing `get SDR` commands with a non-zero value for the offset into the record field must first reserve the SDR repository by issuing a `reserve SDR repository` command.

If you issue a `get SDR` command (storage 23h) with a bytes to read size of FFh (meaning read entire record) this will cause an error in most cases, since SDRs are bigger than the buffer sizes for the typical system interface implementation. The controller therefore returns an error completion code if the number of record bytes exceeds the maximum transfer length for the interface. The completion code CAh indicates that the number of requested bytes cannot be returned. Returning this code is recommended, although a controller could also return an FFh completion code. In either case, the algorithm for handling this situation is to default to using partial reads if the read entire record operation fails (that is, if you get a non-zero completion code).

Table 89 Get SDR command request and response data

Request data byte number	Data field
1	Reservation ID, LS byte. Only required for partial reads with a non-zero offset into record field. Use 0000h for the reservation ID otherwise.
2	Reservation ID, MS byte.
3	Record ID of record being requested, LS byte.
4	Record ID of record being requested, MS byte.
5	Offset into record.
6	Bytes to read. FFh means read entire record.
Response data byte number	Data field
1	Completion code.
2	Record ID for next record, LS byte.
3	Record ID for next record, MS byte.
4:3+N	Record data.

Add SDR command

This command is available to the MC.

This command adds the specified sensor record to the SDR repository and returns its record ID. The data passed in the request must contain the SDR data in its entirety.

Table 90 Add SDR command request and response data

Request data byte number	Data field
1:N	SDR data
Response data byte number	Data field
1	Completion code
2	Record ID for added record, LS byte
3	Record ID for added record, MS byte

Delete SDR command

This command is available to the MC.

This command deletes the sensor record specified by record ID. The requester's ID and the reservation ID must also match the present owner of the SDR repository.

Table 91 Delete SDR command request and response data

Request data byte number	Data field
1	Reservation ID, LS byte
2	Reservation ID, MS byte
3	Record ID of record to delete, LS byte
4	Record ID of record to delete, MS byte
Response data byte number	Data field
1	Completion code
2	Record ID for deleted record, LS byte
3	Record ID for deleted record, MS byte

Clear SDR repository command

This command is available to the MC.

This command clears all records from the SDR repository and re-initializes the SDR repository subsystem. Mainly a development and production aid, use of this command should be avoided in utilities and system management software. The requester's ID and reservation ID information must also match the present owner of the SDR repository.

Table 92 Clear SDR repository command request and response data

Request data byte number	Data field
1	Reservation ID, LS byte
2	Reservation ID, MS byte
3	C (43h)
4	L (4Ch)

Table 92 Clear SDR repository command request and response data *(continued)*

5	R (52h)
6	<ul style="list-style-type: none"> • AAH = initiate erase • 00h = get erasure status
Response data byte number	Data field
1	Completion code
2	Erasure progress <ul style="list-style-type: none"> • [7:4] - reserved • [3:0] - erasure in progress <ul style="list-style-type: none"> ◦ 0h = erasure in progress ◦ 1h = erase completed

Run initialization agent command

This command is available to the MC.

This command can be used to cause the initialization agent to run. The command can be used to check the status of the initialization agent as well.

Table 93 Run initialization agent command request and response data

Request data byte number	Data field
1	<ul style="list-style-type: none"> • [7:1] - reserved • [0] <ul style="list-style-type: none"> ◦ 1b = run initialization agent ◦ 0b = get status of initialization agent process
Response data byte number	Data field
1	Completion code
2	<ul style="list-style-type: none"> • [7:1] - reserved • [0] <ul style="list-style-type: none"> ◦ 1b = initialization completed ◦ 0b = initialization in progress

FRU inventory device commands

The FRU inventory data contains information such as the serial number, part number, asset tag, and a short descriptive string for the FRU. The contents of a FRU inventory record are specified in the Platform Management FRU Information Storage Definition.

The FRU inventory device is a logical device and is not necessarily implemented as a separate physical device. The device that contains the SDR repository device also typically holds FRU inventory information for the main system board and chassis, there may also be a separate FRU inventory device that provides access to the FRU information for a replaceable module such as a memory module.

Get FRU inventory area info command

This command returns the overall size of the FRU inventory area for a device in bytes.

Table 94 Get FRU inventory area info command request and response data

Request data byte number	Data field
1	FRU device id. FFh = reserved
Response data byte number	Data field
1	Completion code.
2	FRU inventory area size in bytes, LS byte.
3	FRU inventory area size in bytes, MS byte.
4	[7:1] — reserved [0] — 0b = device is accessed by bytes, 1b = device is accessed by words

Read FRU data command

This command returns the specified data from the FRU inventory info area. This is effectively a low level direct interface to a non-volatile storage area. This means that the interface does not interpret or check any semantics or formatting for the data being accessed. The offset used in this command is a logical offset that may correspond to the physical address used in the device that provides the non-volatile storage. For example, FRU information kept in flash at physical address 1234h, however the offset 0000h would be used with this command to access the start of the FRU information. IPMI FRU device data (devices formatted per FRU) as well as process and DIMM FRU data always starts from offset 0000h unless otherwise noted.

NOTE: While offset values are 16-bit allowing FRU devices up to 64K words, the count to read, count returned and count written fields are 8-bits. This is in recognition of the limitations on the size of messages. Currently IPMB messages are limited to 32-bytes total.

Table 95 Read FRU data command request and response data

Request data byte number	Data field
1	FRU device id. FFh = reserved.
2	FRU inventory offset to read, LS byte.
3	FRU inventory offset to read, MS byte. Offset is in bytes or words per device access type returned in the get fru inventory area info command.
4	Count to read – count is 1 based.
Response data byte number	Data field
1	Completion code. Generic, plus following command specific: <ul style="list-style-type: none">81h = FRU device busy. The requested cannot be completed because the implementation of the logical FRU device is in a state where the FRU information is temporarily unavailable. This could be due to a condition such as a loss of arbitration if the FRU is implemented as a device on a shared bus.Software can elect to retry the operation after at least 30 milliseconds if this code is returned. NOTE: It is highly recommended that management controllers incorporate built-in retry mechanisms. Generic IPMI software cannot be relied upon to take advantage of this completion code

Table 95 Read FRU data command request and response data *(continued)*

2	Count returned – count is 1 based
3:2+N	Requested data.

Write FRU data command

This command writes the specified byte or word to the FRU inventory info area. This is a low level direct interface to a non-volatile storage area. This means that the interface does not interpret or check any semantics or formatting for the data being written. The offset used in this command is a logical offset that may correspond to the physical address used in device that provides the non-volatile storage. For example, FRU information could be kept in flash at physical address 1234h, however offset 0000h would still be used with this command to access the start of the FRU information. IPMI FRU device data (devices that are formatted per FRU) as well as processor and DIMM FRU data always starts from offset 0000h unless otherwise noted. Updating the FRU inventory data is presumed to be a system level, privileged operation. There is no requirement for devices implementing this command to provide mechanisms for rolling back the FRU inventory area in the case of incomplete or incorrect writes.

Table 96 Write FRU data command request and response data

Request data byte number	Data field
1	FRU device id. FFh = reserved
2	FRU inventory offset to write, LS byte
3	FRU inventory offset to write, MS byte
4:3+N	Data to write
Response data byte number	Data field
1	Completion code. Generic, plus following command specific: <ul style="list-style-type: none"> 80h = write-protected offset. Cannot complete write because one or more bytes of FRU data are to a write-protected offset in the FRU device. An implementation may have allowed a partial write of the data to occur. 81h = FRU device busy. Refer to the preceding table Table 95 (page 125) for the description of this completion code.
2	Count written – count is 1 based

Sensor Device Commands

Get device SDR info command

This command returns general information about the collection of sensors in a dynamic sensor device.

NOTE: Issuing this command without a parameter, returns LUN based device sensor information.

Regarding LUN based device sensor information, a device could implement four sensors under one LUN and twelve under another. SDR info does not return the aggregate of the sensor information, instead you must issue a Get Device SDR Info command for each LUN.

Table 97 Get device SDR info command request and response data

Request data byte number	Data field
1	Operation (optional)

Table 97 Get device SDR info command request and response data *(continued)*

	<p>[7:1] — reserved</p> <p>[0] — 1b = Get SDR count, returns the total number of SDRs in the device.</p> <p>0b = Get sensor count, returns the number of sensors implemented on the LUN addressed.</p>
Response data byte number	Data field
1	Completion code
2	<p>For operation = Get sensor count (or if byte 1 not present in the request): Number of sensors in device for the LUN this command was addressed.</p> <p>For operation = Get SDR count, the total number of SDRs in the device.</p>
3	<p>Flags:</p> <p>Dynamic population</p> <p>[7]</p> <p>0b = static sensor population. The number of sensors handled by this device is fixed and a query shall return records for all sensors.</p> <p>1b = dynamic sensor population. This device may have its sensor population vary during run time (any time other than during installation).</p> <p>Reserved</p> <p>[6:4] — reserved</p> <p>Device LUNs</p> <p>[3] — 1b = LUN 3 has sensors</p> <p>[2] — 1b = LUN 2 has sensors</p> <p>[1] — 1b = LUN 1 has sensors</p> <p>[0] — 1b = LUN 0 has sensors</p>
4:7	<p>Sensor population change indicator. LS byte first.</p> <p>Four byte timestamp, or counter. Updated or incremented each time the sensor population changes. This field is not provided if the flags indicate a static sensor population.</p>

Get device SDR command

The Get Device SDR command allows SDR information for sensors and is typically implemented in a satellite management controller. It also returns SDR types in addition to 01h and 02h. This is an optional command for static sensor devices, and mandatory for dynamic sensor devices. The format action is similar to the get_sdr command for repository devices.

NOTE: A sensor device uses consistent sensor numbers for particular sensor.

The Get Device SDR command includes a reservation_id that notifies the requestor that a record may have changed during a multi-part read.

Table 98 Get device SDR command request and response data

Request data byte number	Data field
1	Reservation ID. LS Byte. Only required for partial reads with a non-zero offset into record field. Use 0000h for reservation id.
2	Reservation ID. MS Byte.
3	Record id of record to get, LS byte. 0000h returns the first record.
4	Record id of record to get, MS byte.

Table 98 Get device SDR command request and response data *(continued)*

5	Offset into record.
6	Bytes to read. FFh means read entire record.
Response data byte number	Data field
1	Completion code. Generic, plus command specific: 80h = record changed. This status is returned if any of the record contents were altered since the last time the requestor issued the request with 00h for the <code>offset into SDR</code> field.
2	Record id for next record, LS byte.
3	Record id for next record, MS byte.
4:3+N	Requested bytes from record.

Reserve device SDR repository command

This command is used to obtain a `reservation id` that is part of the mechanism used to notify the requestor of record changes during a multi-part read.

Table 99 Reserve device SDR repository command request and response data

Response data byte number	Data field
1	Completion code
2	Reservation id, LS byte 0000h reserved.
3	Reservation id, MS byte
Request data byte number	Data field
1	Sensor number (FFh = reserved)

Get sensor thresholds command

This command retrieves the threshold for a given sensor.

Table 100 Get sensor thresholds command response data

Response data byte number	Data field
1	Completion code
2	[7:6] — reserved. Returns as 00b. Readable thresholds: This bit mask indicates which thresholds are readable [5] — 1b = upper non-recoverable threshold [4] — 1b = upper critical threshold [3] — 1b = upper non-critical threshold [2] — 1b = lower non-recoverable threshold [1] — 1b = lower critical threshold [0] — 1b = lower non-critical threshold
3	lower non-critical threshold, if present, ignore on read otherwise
4	lower critical threshold, if present, ignore on read otherwise
5	lower non-recoverable threshold, if present, ignore on read otherwise

Table 100 Get sensor thresholds command response data (*continued*)

Response data byte number	Data field
6	upper non-critical threshold, if present, ignore on read otherwise
7	upper critical, if present, ignore on read otherwise
8	upper non-recoverable, if present, ignore on read otherwise

Get sensor reading command

This command returns the present reading for sensor. The sensor device may return a stored version of a periodically updated reading, or the sensor device may scan to obtain the reading after receiving the request.

The event/reading type code from the SDR determines the state bits returned by discrete sensors.

Table 101 Get sensor reading command request data

Request data byte number	Data field
1	Sensor number (FFh = reserved)
Response data byte number	Data field
1	Completion code.
2	Sensor reading Byte 1: byte of reading. Ignore on read if the sensor does not return a numeric (analog) value.
3	[7] — 0b = All event messages disabled from this sensor. [6] — 0b = sensor scanning disabled. [5] — 1b = reading/state unavailable (formerly: initial update in progress). This bit is set to indicate a re-arm or <code>set event receiver</code> command has been used to request an update of the sensor status, and that update has not occurred yet. Software should use this bit to avoid getting an incorrect status while the first sensor update is in progress. This bit is only required if it is possible for the controller to receive and process a <code>get sensor reading</code> or <code>get sensor event status</code> command for the sensor before the update completes. This is most likely the case for sensors, such as fan RPM sensors that may require seconds to accumulate the first reading after a re-arm. The bit may also indicate when a reading/state is unavailable because the management controller cannot obtain a valid reading or state for the monitored entity, typically because the entity is not present. [4:0] — reserved. Ignore on read.
(4)	For threshold-based sensors: Present threshold comparison status [7:6] — reserved. Returned as 1b. Ignore on read. [5] — 1b = at or above (\geq) upper non-recoverable threshold [4] — 1b = at or above (\geq) upper critical threshold [3] — 1b = at or above (\geq) upper non-critical threshold [2] — 1b = at or below (\leq) lower non-recoverable threshold [1] — 1b = at or below (\leq) lower critical threshold [0] — 1b = at or below (\leq) lower non-critical threshold For discrete reading sensors: [7] = 1b = state 7 asserted [6] = 1b = state 6 asserted [5] = 1b = state 5 asserted [4] = 1b = state 4 asserted

Table 101 Get sensor reading command request data *(continued)*

	[3] = 1 b = state 3 asserted [2] = 1 b = state 2 asserted [1] = 1 b = state 1 asserted [0] = 1 b = state 0 asserted
(5)	For discrete reading sensors only. (Optional) (00h Otherwise) [7] = Reserved. Returned as 1b. Ignore on read. [6] = 1 b = state 14 asserted [5] = 1 b = state 13 asserted [4] = 1 b = state 12 asserted [3] = 1 b = state 11 asserted [2] = 1 b = state 10 asserted [1] = 1 b = state 9 asserted [0] = 1 b = state 8 asserted

DCMI specific commands

This section includes commands specific to the Data Center Manageability Interface implementation in Moonshot. [Table 102 \(page 130\)](#) shows the completion codes and definitions found in data byte 1 of DCMI specific command responses.

Table 102 DCMI completion codes

Code	Definition
00h	Command completed normally.
C0h	Node busy. Command could not be processed because command processing resources are temporarily unavailable.
C1h	Invalid Command. Used to indicate an unrecognized or unsupported command.
C2h	Command invalid for given LUN.
C3h	Timeout while processing command. Response unavailable.
C4h	Out of space. Command could not be completed because of a lack of storage space required to execute the given command operation.
C5h	Reservation cancelled or invalid reservation ID.
C6h	Request data truncated.
C7h	Request data length invalid.
C8h	Request data field length limit exceeded.
C9h	Parameter out of range. One or more parameters in the data field of the request are out of range. This is different from 'Invalid data field' (CCh) code in that it indicates that the erroneous fields have a contiguous range of possible values.
CAh	Cannot return number of requested data bytes.
CBh	Requested sensor, data, or record not present.
CCh	Invalid data field in request.
CDh	Command illegal for specified sensor or record type.
CEh	Command response could not be provided.

Table 102 DCMI completion codes *(continued)*

Code	Definition
CFh	Cannot execute duplicated request. This completion code is for devices which cannot return the response that was returned for the original instance of the request. Such devices should provide separate commands that allow the completion status of the original request to be determined. An event receiver does not use this completion code, but returns the 00h completion code in the response to (valid) duplicated requests.
D0h	Command response could not be provided. SDR repository in update mode.
D1h	Command response could not be provided. Device in firmware update mode.
D2h	Command response could not be provided. MC initialization or initialization agent in progress.
D3h	Destination unavailable. Cannot deliver request to selected destination. For example, this code can be returned if a request message is targeted to SMS, but receive message queue reception is disabled for the particular channel.
D4h	Cannot execute command due to insufficient privilege level or other security-based restriction.
D5h	Cannot execute command. Command, or request parameters, not supported in present state.
D6h	Cannot execute command. Parameter is illegal because command sub-function has been disabled or is unavailable.
FFh	Unspecified error.

Get DCMI capability info command

This command provides version information for DCMI and information about the mandatory and optional DCMI capabilities that are available on the particular platform. The command is session-less, and is a bare-metal provisioning command. The availability of features does not imply the features are configured.

Table 103 Get DCMI capability info command request and response data

Request data byte number	Data field
1	Group extension identification = DCh
2	Parameter selector
Response data byte number	Data field
1	Completion code. See Table 102 (page 130) .
2	Group extension identification = DCh
3:4	DCMI specification conformance: <ul style="list-style-type: none"> • Byte 1 — Major version (01h) • Byte 2— Minor version (01h)
5	Parameter revision = 02h
6:N	Parameter data. See Table 104 (page 132) .

Table 104 DCMI Capabilities Parameters

Parameter	#	Parameter data (non-volatile unless noted)
Supported DCMI capabilities	1	<p>This field returns the supported capabilities available in the server in conformance to DCMI specification for both Platform and Manageability access. All reserved bits shall be set to 0b.</p> <ul style="list-style-type: none"> • Byte 1–Reserved • Byte 2–Platform capabilities. All bits: <ul style="list-style-type: none"> ◦ 0b=Not present ◦ 1b=Available ◦ [7:1] — Reserved ◦ [0] — Power management/monitoring support (Defined as support for either power monitoring or power monitoring plus power limiting.) • Byte 3–Manageability access capabilities. All bits: <ul style="list-style-type: none"> ◦ 0b=Not present ◦ 1b=Available ◦ [7:3] — Reserved ◦ [2] — Out-of-band secondary (second) LAN channel available (optional). ◦ [1] — Serial TMODE available (TMODE on serial port to management controller) (optional) ◦ [0] — Power management/monitoring support (Defined as support for either power monitoring or power monitoring plus power limiting.)
Mandatory platform attributes	2	<p>This field returns the platform attributes for the platform capabilities. All reserved bits shall be set to 0b.</p> <ul style="list-style-type: none"> • Byte 1:2–SEL attributes <ul style="list-style-type: none"> ◦ [15]–SEL automatic rollover enabled (SEL overwrite) (0b=Not present, 1b=available) ◦ [14] Entire SEL Flush upon rollover (Valid if rollover is enabled) (0b=Not present, 1b=available) ◦ [13] Record level SEL flush upon rollover (Valid if rollover is enabled) (0b=Not present, 1b=available) ◦ [12] Reserved ◦ [11:0] Number of SEL entries (maximum of 4096) (the number of entries supported must be 64 or greater to be in conformance) • Byte 3–4–Reserved • Byte 5–Sample frequency for temperature monitoring (in units of 1 second)

Table 104 DCMI Capabilities Parameters *(continued)*

Parameter	#	Parameter data (non-volatile unless noted)
Optional platform attributes	3	<p>This field returns the attributes required for the recommended platform capabilities.</p> <ul style="list-style-type: none"> Byte 1 — Power management device slave address <ul style="list-style-type: none"> [7:1] 7-bit I²C slave address on IPMB. [0] Reserved. Write as 0b. [20h=MC, XXh=Satellite/external controller] Byte 2 — Power management controller channel number <ul style="list-style-type: none"> [7:4] Channel number for channel that management controller is located on. Use 0h for the primary MC. [3:0] Device revision (used for providing the revision control for power management capability)
Manageability access attributes	4	<p>This field returns the attributes of the manageability access.</p> <ul style="list-style-type: none"> Byte 1—Mandatory primary LAN OOB support (RMCP+ support only) [7:0] Channel number (0xFFh==Not supported) Byte 2—Optional secondary LAN OOB support (RMCP+ support only) [7:0] Channel number (0xFFh==Not supported) Byte 3—Optional serial out-of-band TMODE capability [7:0] Channel number (0xFFh==Not supported)

Get asset tag command

This command enables management consoles or local software to get asset tag data. UTF-8 encoding is identified when the first three bytes (offsets 0, 1, and 2) of the returned asset tag data are set to the UTF-8 byte order mark (BOM) pattern, EFh, BBh, BFh, respectively.

Table 105 Get asset tag command request and response data

Request data byte number	Data field
1	Group extension identification = DCh
2	Offset to read
3	<p>Number of bytes to read (16 bytes maximum)</p> <p>NOTE: If the number of bytes to read starting from the given <i>offset to read</i> exceeds the number of remaining asset tag data bytes, the command will complete normally (completion code = 00h) but will only return the remaining bytes (provided the offset to read and bytes to read are within their correct ranges.) For example, if the asset tag length is presently 20 bytes, submitting an offset to read of 16 and a bytes to read of 16 will be accepted, but only the asset tag data bytes at offsets 16–19 will be returned.</p>
Response data byte number	Data field
1	<p>Completion code. C9h shall be returned if offset >62, offset to read+bytes to read >63, or bytes to read is > 16.</p> <p>The following applies to implementations that keep the DCMI asset tag and IPMI FRU asset tag information synchronized:</p> <p>If the encoding indicated by the Type/Length byte in the IPMI FRU is not set to ASCII+Latin1 or the language code for the IPMI FRU product info area is not set to English (0 or 25), the command</p>

Table 105 Get asset tag command request and response data *(continued)*

	shall return the requested data bytes, but shall also return a command-specific completion code based on the detected encoding type, as follows: <ul style="list-style-type: none"> • 80h=Encoding type in FRU is binary/unspecified • 81h=Encoding type in FRU is BCD Plus • 82h=Encoding type is FRU is 6-bit ASCII Packed • 83h=Encoding type is set to ASCII+Latin1 but language code is not set to English (indicating data is 2-byte Unicode). The management controller does not check, nor require, a BOM in the asset tag data; asset tag data can be stored and retrieved as ASCII+Latin1 without receiving an error completion code.
2	Group extension identification = DCh
3	Total asset tag length (must be less than or equal to 64 bytes)
4:N	Asset tag data (starting from the offset to read)

Get DCMI sensor info command

This DCMI command returns information about a DCMI-specified sensor. A particular sensor is identified by the combination of its entity ID and entity instance numbers.

Table 106 DCMI Entity ID extension

Entity ID description	Entity ID	Entity instance	Sensor type
Inlet temperature	40h	0x01...n	Temp (01h)
CPU temperature (based on number of processors or cores)	41h	0x01...n	Temp (01h)
Baseboard temperature	42h	0x01...n	Temp (01h)

Table 107 Get DCMI sensor info command request and response data

Request data byte number	Data field
1	Group extension identification = DCh
2	Sensor type
3	Entity ID
4	Entity instance <ul style="list-style-type: none"> • 00h = Retrieve information about all instances associate with entity ID • 01h-FFh = Retrieve only the information about particular instance
5	Entity instance start, used with entity instance 00h for number of instance exceeding one IPMI response.
Response data byte number	Data field
1	Completion code. See Table 102 (page 130) .
2	Group extension identification = DCh
3	Total number of available instances for the entity ID.

Table 107 Get DCMI sensor info command request and response data *(continued)*

4	Number of record IDs in this response (maximum of 8 per response) 01h for entity instance not equal to 00h.
5:6 + N	<p>SDR record ID corresponding to the entity IDs.</p> <ul style="list-style-type: none"> Byte 1: Record ID LS byte, used for retrieving SDR records. Byte 2: Record ID MS byte, used for retrieving SDR records. <p>NOTE: The management controller can include SDR record IDs corresponding to entities IDs compatible IPMI 2.0 specified entity IDs such as:</p> <ul style="list-style-type: none"> Request for inlet temp (40h) can also include air inlet temp info (37h) Request for CPU temp (41h) can also include CPU temp (03h) Request for baseboard temp (42h) can also include system board (07h)

Set asset tag command

This command enables remote consoles or local software to set the asset tag data. UTF-8 encoding is identified when the first three bytes (offsets 0, 1, and 2) of the returned asset tag data are set to the UTF-8 byte order mark (BOM) pattern, EFh, BBh, BFh, respectively. Otherwise the data encoding is assumed to be the ASCII+Latin1 subset. Note that the management controller simply stores all eight bits of each of the given asset tag data bytes. It does not check the encoding of the asset tag data bytes, nor does it check for a BOM in the data.

Implementations that keep the asset tag in synch with the IPMI FRU data shall write the given characters to the asset tag field in the product info area of the IPMI FRU device and set the encoding of the corresponding type/length byte field to ASCII+Latin1.

Table 108 Set asset tag command request and response data

Request data byte number	Data field
1	Group extension identification = DCh
2	Offset to write (0 to 62). The offset is relative to the first character of the asset tag data.
3	<p>Number of bytes to write (16 bytes maximum)</p> <p>NOTE: The command must set the overall length of the asset tag (in bytes) to the value (offset to write + bytes to write). Any pre-existing asset tag bytes at offsets past that length are automatically deleted.</p>
4–N	Asset tag data
Response data byte number	Data field
1	<p>Completion code. C9h shall be returned if offset >62, offset+bytes to write >63, or bytes to write >16.</p> <p>A C9h completion code shall also be returned if an attempt is made to write to an offset that is more than one greater than the length of the presently stored asset tag data. Set operations for asset tags must be contiguous. For example, if the asset tag is presently seven bytes long an attempt to write starting at offset 10 will be rejected and a C9h completion code returned.</p>
2	Group extension identification = DCh
3	<p>Total asset tag length. This is the length in bytes of the stored asset tag after set operation has completed. The asset tag length shall be set to the sum of the offset to write plus bytes to write. For example, if offset to write is 32 and bytes to write is 4, the total asset tag length returned will be 36.</p>

Management controller ID string

The management controller ID string is provided in order to accommodate the requirement for the management controllers to identify themselves during discovery phases. Set/get management

controller identifier string commands are provided to provision the controller with the unique identification. The management controller must maintain the management controller identifier string as non-volatile data.

The management controller ID string is used to override the default OEM provided ID for DHCP discovery. If the management controller ID string is not provisioned, then the default controller ID shall be "DCMI<mac-address>". The maximum length of the identifier string shall be 64 bytes including a NULL terminator.

Get controller ID string command

Table 109 Get controller ID string command request and response data

Request data byte number	Data field
1	Group extension identification = DCh
2	Offset to read
3	Number of bytes to read (16 bytes maximum)
Response data byte number	Data field
1	Completion code. See Table 102 (page 130) .
2	Group extension identification = DCh
3	Total length. NOTE: The maximum length of the identifier string must not exceed 64 bytes.
4–N	Data

Set controller ID string command

Table 110 Set controller ID string command request and response data

Request data byte number	Data field
1	Group extension identification = DCh
2	Offset to write
3	Number of bytes to write (16 bytes maximum)
4–N	Data
Response data byte number	Data field
1	Completion code. See Table 102 (page 130) .
2	Group extension identification = DCh
3	Total length written. NOTE: The maximum length of the identifier string must not exceed 64 bytes.

PICMG specific commands

Get PICMG properties command

Table 111 Get PICMG properties command request and response data

Request data byte number	Data field
1	<i>PICMG Identifier</i> . Indicates that this is a PICMG-defined group extension command. A value of 00h is used.
Response data byte number	Data field
1	Completion code.
2	<i>PICMG Identifier</i> . Indicates that this is a PICMG-defined group extension command. A value of 00h is used.
3	<i>PICMG Extension Version</i> . Indicates the version of PICMG extensions implemented by the IPM Controller. [7:4] BCD encoded minor version [3:0] BCD encoded major version This specification defines version 2.3 of the PICMG extensions. IPM Controllers implementing the extensions as defined by this specification report a value of 23h. The value 00h is reserved.
4	<i>Max FRU Device ID</i> . The numerically largest FRU Device ID for the Managed FRUs implemented by this IPM Controller, excluding the following FRU Device IDs reserved at the top of the range for special purposes: <ul style="list-style-type: none">• 254 — This FRU Device ID does not represent any Managed FRU and is used only certain commands directed at the Shelf Manager for accessing logical Shelf FRU information.• 255 — Reserved by IPMI specification.
5	<i>FRU Device ID for IPM Controller</i> . Indicates a FRU Device ID for the FRU containing the IPM Controller. IPM Controllers report zero (0).

Get address info command

Table 112 Get address info command request and response data

Request data byte number	Data field
1	<i>PICMG identifier</i> . Indicates that this is a PICMG-defined group extension command. A value of 00h is used.
2	<i>FRU Device ID</i> . Indicates an individual FRU site, which must identify a FRU being managed by the IPM Controller. This field is optional. If this field is not present, the command will return addressing information for the FRU containing the IPM Controller that implements the command. This field is ignored when Address Key Type is set to Physical Address. This field is required if Address Key Type is present.
3	<i>Address Key Type</i> . This field defines the type of address that is being provided in the Address Key field; only Physical Address is defined when this command is addressed to an IPM Controller. This field is optional. If this field is not present, the command will return addressing information for the FRU specified by the FRU Device ID. 03h = Physical Address All other values reserved.
4	<i>Address Key</i> . This field is required if Address Key Type is present, and holds the Site Number of a FRU managed by the IPM Controller in this extension.
5	<i>Site Type</i> . This field is required if Address Key Type is Physical Address.
Response data byte number	

Table 112 Get address info command request and response data *(continued)*

1	Completion code.
2	PICMG identifier. Indicates that this is a PICMG-defined group extension command. A value of 00h is used.
3	<i>Hardware Address</i> . Indicates the Hardware Address of the IPM Controller.
4	<i>IPMB-0 Address</i> . Indicates the IPMB address for IPMB-0 of the IPM Controller.
5	<i>Reserved</i> . Has a value FFh. Other values reserved in PICMG 2.9.
6	<i>FRU Device ID</i> . The FRU Device ID associated with the FRU that the Site Number designates.
7	<i>Site Number</i> . The Site Number associated with the FRU that the FRU Device ID designates. This Site Number is IPM Controller relative except for AdvancedTCA Boards or Rear Transition Modules. If the IPM Controller does not provide this information, it must return 0.
8	<i>Site Type</i> . See Table 113 (page 138) .
9	<i>Reserved</i> . Has a value FFh. Other values reserved in PICMG MTCA.0.
10	<i>Address on IPMI Channel 7</i> . Specifies the address of the management controller representing the FRU on IPMI Channel Number 7, if such controller exists and is accessible through message bridging. If there is no such controller or if it is not IPMI-compatible, this byte is not returned or contains a value of FFh.

Table 113 Site type values

Site type description	Site type
Front board	00h
Power entry	01h
Shelf FRU information	02h
Dedicated ShMC	03h
Fan tray	04h
Fan filter tray	05h
Alarm	06h
Advanced MC module	07h
PMC	08h
Rear transition module. ¹	09h
Reserved	0Ah-BFh
OEM	C0h-CFh
Reserved	D0h-FEh
Unknown	FFh

¹ Also applies to front transition modules in ATCA300 shelves.

FRU inventory device lock control command

Table 114 FRU inventory device lock control command request and response data

Request data byte number	Data field
1	PICMG identifier. Indicates that this is a PICMG-defined group extension command. A value of 00h is used.
2	FRU device ID. Indicates the FRU inventory device to be locked. This value must be 254.

Table 114 FRU inventory device lock control command request and response data *(continued)*

3	<p>Operation. The operation to perform on the FRU lock.</p> <p>0 Get last commit timestamp. Used to fetch the FRU inventory device's last commit timestamp.</p> <p>1 Lock. Lock the FRU inventory device lock and return a lock ID.</p> <p>2 Unlock and discard. Unlock the FRU inventory device lock and discard all writes since the last lock.</p> <p>3 Unlock and commit. Unlock the FRU inventory device lock and commit all writes since the last lock.</p>
4–5	FRU inventory device lock ID. For operations 2 and 3 above, this is the lock ID to unlock. The value must be zero for other operations.
Response data byte number	Data field
1	<p>Completion code:</p> <p>80h Invalid FRU information. An unlock and commit operation was requested and the FRU information was invalid.</p> <p>81h Lock failed. A lock operation was requested and the FRU inventory device was already locked, or an unlock operation was requested and the FRU inventory device lock ID was invalid.</p>
2	PICMG identifier. Indicates that this is a PICMG-defined group extension command. A value of 00h is used.
3–4	FRU inventory device lock ID. For the lock operation, this returns the FRU inventory device lock ID for the lock if the lock operation was successful. This value is chosen by the MC arbitrarily, but is chosen in such a way to avoid reusing a recently used number. For all other operations this value is set to zero.
5–8	FRU inventory device last commit timestamp. This value is the timestamp of the last successful commit operation on the FRU inventory device.

6 IPMI Messaging and Interfaces

IPMI uses message based interfaces for the different interfaces to the platform management subsystem such as IPMB, LAN, and the system interface to the MC.

All IPMI messages share the same fields in the message payload regardless of the interface (transport) that they are transferred over. The same core of IPMI messages is available over every IPMI specified interface, just wrapped differently according to the needs of the particular transport. This enables management software that works on one interface to be converted to use a different interface by changing the underlying driver for that particular transport. This also enables knowledge reuse, that is a developer who understands the operation of IPMI commands over one interface can readily apply that knowledge to a different IPMI interface.

IPMI messaging uses a request/response protocol. IPMI request messages are commonly referred to as commands. The use of a request/response protocol facilitates the transfer of IPMI messages over different transports. It also facilitates multi-master operations on busses like the IPMB, allowing messages to be interleaved and multiple management controllers to directly intercommunicate on the bus.

IPMI commands are grouped into functional command sets, using a field called the network function code (NetFn). There are command sets for sensor and event related commands, chassis commands, and others. This functional grouping makes it easier to organize and manage the assignment and allocation of command values.

All IPMI request messages have a network function command and optional data fields. All IPMI response messages carry network function command optional data and a completion code field. As inferred earlier, the differences between the different interfaces has to do with the framing and protocols used to transfer this payload. For example, the IPMB protocol adds fields for I²C and controller addressing and data integrity checking and handling whereas the LAN interface adds formatting for sending IPMI messages as LAN packets.

System Interfaces

IPMI defines three standard system interfaces that system software use for transferring IPMI messages to the MC (of which Moonshot supports two). In order to support a variety of microcontrollers, IPMI offers a choice of system interface, this is also key to enabling cross-platform software. The system interfaces are similar enough so that a single driver can be created that supports all IPMI system interfaces.

The system interface connects to a system bus that can be driven by the main processor(s). The present IPMI system interfaces can be I/O or memory mapped. Any system bus that allows the main processor(s) to access the specified I/O or memory locations, and meet the timing specifications, can be used. Thus, an IPMI system interface could be hooked to the X-bus, PCI, LPC or a proprietary bus off the baseboard chipset.

IPMI system interfaces:

- Keyboard controller style (KCS) — the bit definitions and operation of the registers follow that used in the Intel 8742 Universal Peripheral Interface microcontroller. KCS reflects the fact that the 8742 interface was used as the legacy keyboard controller interface in PC architecture computer systems. This interface is available built-in to several commercially available microcontrollers. Data is transferred across the KCS interface using a per byte handshake.
- SMBus System Interface (SSIF) — This interface is a low pin count option that specifies accessing an MC that is connected to the system SMBus host controller. SSIF helps support lower cost MC implementations by enabling an interface that can be used on low cost microcontrollers in low pin count packages.

NOTE: The SSIF typically has a much lower bandwidth to the MC than the other system interfaces owing to the 100 kbps maximum data rate presently specified for SM Bus.

Message interface description

The heart of this specification is the definition of the messages and data formats used for implementing sensors, event messages, event generators and event receivers, the SDR Repository, and the SEL in the platform management architecture. These messages are designed for delivery using a messaging interface with a particular set of characteristics. This section presents the general specification of that interface, and the messages.

The message interface is defined as a request/response interface. That is, a request message is used to initiate an action or set data, and a response message is returned to the requester. In this document, request messages are often referred to as commands, and response messages as responses.

All messages in this specification share the same common elements as the payload to the command interpreter in the logical device that receives the message. The messaging interfaces differ in the framing, physical addressing, and data integrity mechanisms that are used to deliver the payload.

Table 115 Common message components

Component	Description
Network Function (NetFn)	A field that identifies the functional class of the message. The network function clusters IPMI commands into different sets.
Request/Response identifier	A field that unambiguously differentiates request messages from response messages. In the IPMB protocol, this identifier is merged with the NetFn code where even numbered network function codes identify request messages and odd numbered network function codes identify response messages.
Requester's ID	Information that identifies the source of the request. This information must be sufficient to allow the response to be returned to the correct requestor. For example, the IPMB requesters ID consists of the slave address and LUN of the requester device. For a multiple stream system interface the requesters ID is the stream ID for the stream through which the request was issued.
Responder's ID	A field that identifies the Responder to the request. In request messages this field is used to address the request to the desired responder, in response messages this field is used to assist the requester in matching up a response with a given request.
Command	The messages specified in this document contain a one-byte command field. Commands are unique within a given network function. Command values can range from 00h through FDh. Code FEh is reserved for future extension of the specification and code FFh is reserved for message interface level error reporting on potential future interfaces.
Data	The data field carries the additional parameters for a request or a response, if any.

IPMI Messaging Interfaces

In Moonshot, there are two system interface implementations specified for the MC: KCS and SSIF. The MC can also be reached through additional interfaces such as the IPMB and LAN interfaces.

Network function codes

The network layer in the connection header includes a six-bit field identifying the function accessed. The remaining two bits are the LUN field. The LUN field provides further sub-addressing within the node.

The network function clusters commands into functional command sets. In a parsing hierarchy, the LUN field may be thought of as the selector for a particular network function handler in the node, and the network function may be considered the selector for a particular command set handler within the node.

[“Moonshot network function codes” \(page 142\)](#) defines the supported network functions. With the exception of the application and firmware transfer network functions, the commands and responses

for a given network function are not node specific. The format and function for standard command sets is specified later.

Table 116 Moonshot network function codes

Value(s)	Name	Meaning	Description
00, 01	Chassis	Chassis device requests and responses	00h identifies the message as a command/request and 01h as a response, relating to the common chassis control and status functions.
04, 05	Sensor/Event	Sensor and event requests and responses	This functionality can be present on any node. 04h identifies the message as a command/request and 05h as a response, relating to the configuration and transmission of event messages and system sensors. In Moonshot, this includes event and sensor device commands.
06, 07	Application	Application requests and responses	06h identifies the message as an application command/request and 07h a response. The exact format of application messages are implementation-specific for a particular device, with the exception of App messages that are defined by the IPMI specifications. In moonshot, this includes IPMD global commands, MC watchdog timer commands, and MC device and messaging commands.
0A, 0B	Storage	Non-volatile storage requests and responses	This functionality can be present on any node that provides non-volatile storage and retrieval services. In Moonshot, this includes FRU device, SDR device and SEL device commands.
0C, 0D	Transport	Media-specific configuration & control	Requests (0Ch) and responses (0Dh) for IPMI-specified messages that are media-specific configuration and operation, such as configuration of serial and LAN interfaces. In Moonshot, this includes LAN device and serial device commands.
2Ch–2Dh	Group extension	Non-IPMI group requests and responses	The first data byte position in requests and responses under this network function identifies the defining body that specifies command functionality. Software assumes that the command and completion code field positions will hold command and completion code values. The following values are used to identify the defining body. <ul style="list-style-type: none"> • 00h PICMG — PCI Industrial Computer Manufacturer's Group at http://www.picmg.com • DCh DCMI specifications at http://www.intel.com/go/dcmi • All other reserved Moonshot supports PICMG and DCMI group extensions only. When this network function is used, the ID for the defining body occupies the first data byte in a request, and the second data byte (following the completion code) in a response.

Completion codes

All response messages specified in this document include a completion code as the first byte in the data field of the response. A management controller that gets a request to an invalid (unimplemented) LUN must return an error completion code using that LUN as the responder's LUN (RsLUN) in the response. The completion code indicates whether the associate request messages

completed successfully and normally, and if not, provides a value indicating the completion condition.

Completion codes work at the command level. They are responses to the interpretation of the command after it has been received and validated through the messaging interface. Errors at the network (messaging interface) level are handled with a different error reporting mechanism.

Completion code values are split into generic, device-specific (which covers OEM) and command-specific ranges. All commands can return generic completion codes. Commands that complete successfully return the 00h command completed normally, completion code. Commands that produce error conditions or return a response that varied from what was specified by the request parameters for the command, return a non-zero completion code as specified in [“Completion Codes” \(page 143\)](#).

Table 117 Completion Codes

Code	Definition
GENERIC COMPLETION CODES 00h, C0h-FFh	
00h	Command completed normally
C0h	Node Busy. Command could not be processed— command processing resources temporarily unavailable.
C1h	Invalid Command. Indicates an unrecognized or unsupported command.
C2h	Command invalid for given LUN.
C3h	Timeout while processing command. Response unavailable.
C4h	Out of Space. Command could not be completed because of a lack of storage space.
C5h	Reservation canceled or invalid reservation id.
C6h	Request data truncated.
C7h	Invalid request data length.
C8h	Request data field length limit exceeded.
C9h	Parameter out of range. One or more parameters in the data field of the Request are out of range. This is different from Invalid data field (CCh) code in that it indicates that the erroneous field(s) has a contiguous range of possible values.
CAh	Cannot return number of requested data bytes.
CBh	Requested sensor, data, or record not present.
CCh	Invalid data field in request.
CDh	Command illegal for specified sensor or record type.
CEh	Command response could not be provided.
CFh	Cannot execute duplicated request. This completion code is for devices which cannot return the response that was returned for the original instance of the request. Such devices should provide separate commands that allow the completion status of the original request to be determined.
D0h	Command response could not be provided. SDR repository in update mode.
D1h	Command response could not be provided. Device in firmware update mode.
D2h	Command response could not be provided. MC initialization or initialization agent in progress.
D3h	Destination unavailable. Cannot deliver request to selected destination. Example, this code can be returned if a request message is targeted to SMS, but receive message queue reception is disabled for the particular channel.

Table 117 Completion Codes *(continued)*

Code	Definition
D4h	Cannot execute command due to insufficient privilege level or other security based restriction (example, disabled for firmware firewall).
D5h	Cannot execute command. Command or request parameter(s) not supported in present state.
D6h	Cannot execute command. Parameter is illegal because command sub-function has been disabled or is unavailable (example, disabled for firmware firewall).
FFh	Unspecified error.
DEVICE SPECIFIC (OEM) CODESs 01h — 7Eh	
01h — 7Eh	Device specific (OEM) completion codes. This range is used for command specific codes that are also specific for a particular device and version. A priori knowledge of the device command set is required for interpretation of these codes.
COMMAND SPECIFIC CODES 80h — BEh	
80h — BEh	Standard command specific codes. This range is reserved for command specific completion codes for commands specified in this document.
all other	Reserved.

Additional command specific completion codes, if any, are listed in the completion code field description for the command. In some cases, use of certain command specific completion codes is mandatory. This will be listed alongside the description of the completion code in the command table. If no command specific completion codes are listed, the description will solely indicate that the field is the completion code field.

NOTE: The generic completion code values can be used with any command, regardless of whether additional command specific completion codes are defined.

Channel Model, Authentication, Sessions, and Users

IPMI v2.0 incorporates a common communication infrastructure referred to as the Channel Model.

Channels provide the mechanism for directing the routing of IPMI messages between different media connections to the MC. A channel number identifies a particular connection. For example, 0 is the channel number for the primary IPMB. Up to nine total channels can be supported (the System Interface and primary IPMB, plus seven additional channels with a media type assigned by the implementer.) Channels can thus be used to support multiple IPMB, LAN, Serial, etc., connections to the MC.

Channels can be session-based or session-less. A session is used for two purposes:

1. As a framework for user authentication.
2. To support multiple IPMI messaging streams on a single channel.

Session-based channels thus have at least one user login and support user and message authentication. Session-less channels do not have users or authentication. A LAN channel is session-based, while the System Interface and IPMB are examples of session-less channels.

In order to do IPMI messaging using a session, a session must first be activated. Activating a session authenticates a particular user.

A session has a *Session ID* that is used for tracking the state of a session. The *Session ID* mechanism allows multiple sessions to be simultaneously supported on a channel.

The concept of user is essentially a way to identify a collection of privilege and authentication information. User configuration is done on a per channel basis. This means that a given user could have a different password and set of privileges for accessing the MC via a LAN channel than via a serial channel.

Privilege Levels determine which IPMI commands a given user can execute over a given channel. Privilege Limits set the maximum privilege level at which a user can operate. A user is configured with a given maximum privilege limit for each channel. In addition, there is a *Channel Privilege Limit* that sets the maximum limit for all users on a given channel. The *Channel Privilege Limit* takes precedence over the privilege configured for the user. Thus, a user can operate at a privilege level that is no higher than the lower of the *User Privilege Limit* and the *Channel Privilege Limit*.

Channel numbers

Each interface has a channel number that is used when configuring the channel and for routing messages between channels. Only the channel number assignments for the primary IPMB and the System Interface are fixed, the assignment of other channel numbers can vary on a per-platform basis. Software uses a `Get Channel Info` command to determine what types of channels are available and what channel number assignments are used on a given platform. The following table describes the assignment and use of the channel numbers:

Table 118 Moonshot channel number assignments

Channel Number	Type/Protocol	Description
0h	Primary IPMB	Assigned for communication with the primary IPMB. IPMB protocols are used for IPMI messages.
2h	LAN	Assigned for communication between the zone MC and the LAN. RMCP+ is used as the protocol.
7h	IPMB_L	Secondary IPMB, used for communication between the cartridge and system node controllers. IPMB protocols are used.
Eh	Present channel	The value Eh is used as a way to identify the current channel that the command is being received from, for example, if software wants to know what channel number it's presently communicating over, it can find out by issuing a <code>Get Channel Info</code> command for channel E.
Fh	System interface	Assigned for routing messages to the system interface.

Logical channels

From the IPMI Messaging point-of-view, a party that bridges a message from one channel to another only is mainly concerned that it gets the correct response from the MC. Often, it does not matter to remote console or system software whether the target channel and devices are physically implemented or not. For example, in Moonshot the IPMB and IPMB-L are logical channels.

Channel Privilege Levels

Channel privilege limits determine the maximum privilege that a user can have on a given channel. One channel can be configured to allow users to have up to Administrator level privilege, while another channel may be restricted to allow no higher than User level. The privilege level limits take precedence over the privilege level capabilities assigned per user.

Channels can be configured to operate with a particular maximum Privilege Level. Privilege levels tell the MC which commands are allowed to be executed via the channel. The `Set Channel Access` command sets the maximum privilege level limit for a channel. The `Set Session Privilege Level` command requests the ability to perform operations at a particular privilege level. The `Set Session Privilege Level` command can only be used to set privilege levels that are less than or equal to the privilege level limit for the entire channel, regardless of the privilege level of the user.

Table 119 Channel privilege levels

Channel privilege level	Description
Callback	Lowest privilege level. Only commands necessary to support initiating a <code>Callback</code> are allowed.
User	Primarily commands that read data structures and retrieve status. Commands that can be used to alter MC configuration, write data to the management controllers, or perform system actions such as resets, power on/off, and watchdog activation are disallowed.
Operator	All MC commands are allowed, except for configuration commands that can change the behavior of the out-of-band interfaces. For example, operator privilege does not allow the capability to disable individual channels or higher.
Administrator	All MC commands are allowed, including configuration commands. An administrator can even execute configuration commands that would disable the channel that the administrator is communicating over.

Users & Password support

User in this specification refers to a collection of data that identifies a password for establishing an authenticated session and the privileges associated with that password. For configuration purposes, sets of user information are organized and accessed according to a numeric *User ID*. When activating a session, user information is looked up with a text *username*.

NOTE: In Moonshot and HP iLO, anonymous users are not supported due to security concerns.

User access can be enabled on a per channel basis. Thus, different channels can have different sets of users enabled.

If desired, a username on one channel can be associated with a different password than the same username on a different channel. When a session is activated the MC scans usernames sequentially starting with User ID 1 and looks for the first user with matching username and access granted for the given channel. Thus, having different passwords for a given username requires configuring multiple user entities — one for each different password being used for a particular set of channels.

The specification allows a number of different implementations for supporting users on a channel. Minimum requirements include:

- No anonymous user access.
- User names may be fixed or configured, or a combination of both, at the choice of the implementation.
- Support for configuring user passwords for all User IDs is required.
- Support for setting per user privilege limits is optional. If the `Set User Access` command is not supported, the privilege limits for the channel are used for all users.

IPMI sessions

Authenticated IPMI communication to the MC is accomplished by establishing a session. Once established, a session is identified by a Session ID. The Session ID may be thought of as a handle that identifies a connection between a given remote user and the MC, using either the LAN or Serial/Modem connection.

The specification supports having multiple active sessions established with the MC. Moonshot supports up to 58 simultaneous sessions at the zone.

The specification also allows a given endpoint (identified by an IP address) on the LAN to open more than one session to a MC. This capability allows a single system to serve as a proxy providing MC LAN sessions for other systems. It is not intended for one system to use this provision to open multiple session to the MC for that systems sole use.

An IPMI messaging connection to the MC fits one of three classifications, session-less, single-session, or multi-session.

Session-less connections

A session-less connection is unauthenticated. There is no user login required for performing IPMI messaging. The System Interface and IPMB are examples of session-less connections.

A special case of a session-less connection can occur over an interface that supports session-based messaging. Session-based connections have certain commands that are accepted and responded to outside of a session. When that occurs, the channel is effectively operating in a session-less manner for those commands. Commands that are handled outside of a session have fixed values for session-specific fields in the message. For example, when the `Get Channel Authentication Capabilities` is sent over a LAN channel outside of a session, the Session ID is set to NULL and authentication type set to NONE in the IPMI session header. Note that commands accepted outside of a session can also be accepted within the context of a session, in which case they must have valid Session IDs, etc. in the session header to be accepted.

Session inactivity timeouts

A session is automatically closed if no new, valid message has been received for the session within the specified interval since the last message. The session must be re-authenticated to be restored. A remote console can optionally use the `Activate Session` command to keep a session open during periods of inactivity.

Note that only an active session will keep the `Session Inactivity Timeout` from expiring. IPMI message activity that occurs outside of an active session has no effect. This is to prevent someone from keeping a phone connection indefinitely while trying to guess different passwords to activate a session.

The MC only monitors for inactivity while the connection is switched over to the MC. Note that closing a session is not always the same as hanging up a modem connection. Serial/modem sessions are also automatically closed when the connection is switched over to the system, but the phone connection remains active. The MC only terminates the phone connection if a session is closed due to an inactivity timeout while the serial connection is routed to the MC.

The timeout and tolerance values are specified for the MC that will timeout and close the session. System software should take this tolerance into account, plus any additional delays due to media transmission times, etc.

An implementation can provide an option to allow timeout configuration via a parameter in the configuration parameters for the given channel type.

System interface messaging

Messaging between system software and the other management BUSes such as the IPMB, is accomplished using channels and a `Receive Message Queue`. A channel is a path through the MC that allows messages to be sent between the system interface and a given bus or message transport. The `Receive Message Queue` is used to hold message data for system software until system software can collect it. All channels share the `Receive Message Queue` for transferring messages to system management software. The `Receive Message Queue` data contains channel, session, and IPMI addressing information that allows system software to identify the source of the message, and to format a message back to the source if necessary.

System management software is responsible for emptying the `Receive Message Queue` whenever it has data in it. Messages are rejected if the `Receive Message Queue` gets full. It is recommended that the `Receive Message Queue` have at least two slots for each channel. The `Receive Message Queue` is a logical concept. An implementation may choose to implement it as an actual queue, or could implement separate internal buffers for each channel. It is recommended that the implementation attempt to leave a slot open for each channel that does not presently have a

message in the queue. This helps prevent lockout by having the queue fill with just messages from one interface.

The MC itself can, if necessary, use the `Receive Message Queue` and `Messaging Channels` to send asynchronous messages to system management software. The recommended mechanism for accomplishing this is to define a unique channel with a protocol type of system. To send an asynchronous message to system software the MC would place a message from that channel directly into the `Receive Message Queue` in System format. System software would be able to respond back to the MC using a `Send Message` command for that channel.

Bridging

MC Messaging Bridging provides a mechanism for routing IPMI Messages between different media. Bridging is only specified for delivering messages between channels; it is not specified for delivering messages between two sessions on the same channel.

With IPMI 2.0, bridging is extended to support delivering IPMI messages between active connections/sessions on the same channel.

There are three mechanisms for bridging messages between different media connected to the MC, depending on the message target:

- MC LUN 10b — used for delivering messages to the System Interface. The MC automatically routes any messages it receives via LUN 10b to the `Receive Message Queue`.
- `Send Message` command from System Interface — used for delivering messages to other channels, such as the IPMB. The messages appear on the channel as if they've come from MC LUN 10b. Thus, if the message is a request message, the response goes to MC LUN 10b and the MC automatically places the response into the `Receive Message Queue` for retrieval. System software is responsible for matching the response up with the original request, thus the `No Tracking` setting in the `Send Message` command.
- `Send Message` command with response tracking — used with response tracking for bridging request messages to all other channels except when the System Interface is the source or destination of the message.

MC LUN 10b

Messages to SMS are always routed to the `Receive Message Queue` and the `Send Message` command is not typically used. Messages to SMS are delivered via the MC SMS LUN 10b. The MC automatically reformats and places any messages that are addressed to LUN 10b into the `Receive Message Queue` for SMS to retrieve using the `Get Message` command.

Sending a request to SMS requires formatting the command so that it is address to MC LUN 10b. SMS can retrieve the request from the `Receive Message Queue`, extract the originator's address and channel info, and then use the `Send Message` command to deliver a response.

The MC does not track requests and responses for messages to system software because the `Receive Message Queue` provides the channel and session information necessary to format the `Send Message` command to deliver the response. System software is capable of tracking the channel and session information it used when generating a request. The `No Tracking` option is used for `Send Message` commands from system software.

The responder then delivers its response message to MC LUN 10b and the response gets routed to the `Receive Message Queue`. Conversely, if a channel wants to deliver a message to SMS, it sends the request message to MC LUN 10b, and later SMS uses a `Send Message` command to return the response from MC LUN 10b.

Send Message command with response tracking

The `Send Message` command is used primarily to direct the MC to act as a proxy that translates a message from one IPMI messaging protocol to another. The MC formats the data for the target channel type and protocol and delivers it to the selected medium.

Media such as the IPMB do not include channel number and session information as part of their addressing information. As a result, request messages from another channel must be delivered as if they originated from the MC itself.

If the bridged message is a request, it is necessary for the MC to hold onto certain data, such as originating channel and session information, so that when the response message comes back it can reformat the response and forward it back to the originator of the request. The primary way the MC accomplishes this is by assigning a unique sequence number to each request that it generates, and saving a set of information in a Pending Bridged Response table that is later used to reformat and route a response back to the originator of the request.

The sequence number returned in the response is used to look up who generated the original response, the saved formatting and address information. The MC then reformats and delivers the response to the original requester and deletes the request from its list of pending responses. The `Send Message` command includes a parameter that directs the MC to save translation information for and track outstanding request messages for the purpose of routing the response back to the originator of the `Send Message` command.

NOTE: With the exception of messages to SMS, when the `Send Message` command is used to deliver a message to a given medium the message appears to have been originated by the MC. This means that a controller on the IPMB can't generically distinguish a bridged request from SMS from a bridged request from LAN.

Table 120 Message bridging mechanism by source and destination

Message Type and direction	Delivery Mechanism	MC tracks pending responses
Request or Response from system interface to any other channel	Send Message	no
Request or Response to system interface from any other channel	MC LUN 10b	no
Request from any channel except system interface to IPMB	Send Message	yes
Response from IPMB to any channel except system interface	MC LUN 00b	yes

Bridged Request Example

This example illustrates a `Send Message` command from the LAN being used to deliver a request to IPMB.

The MC uses the sequence number that it places on the bridged request to identify the channel where the request came from and where to send the response. It is important for the MC to ensure that unique sequence numbers are used for pending requests from each channel. It is also important that sequence numbers are unique for successive requests to a given responder. One way to manage sequence numbers to the IPMB is to track them on a per responder basis. This can be kept in a table of Pending Bridged Response information.

In order to get the response back to the LAN, the IPMB response must return the same sequence number that was passed in the request. The management controller uses the sequence number to look up the channel type specific addressing, sequence number, and security information that it stored when the request was forwarded. For example, if the channel type is LAN then the response message must be formatted up in an RMCP/UDP packet with the IP address of the requester, the sequence number passed in the original request, the appropriate security key information, and so on.

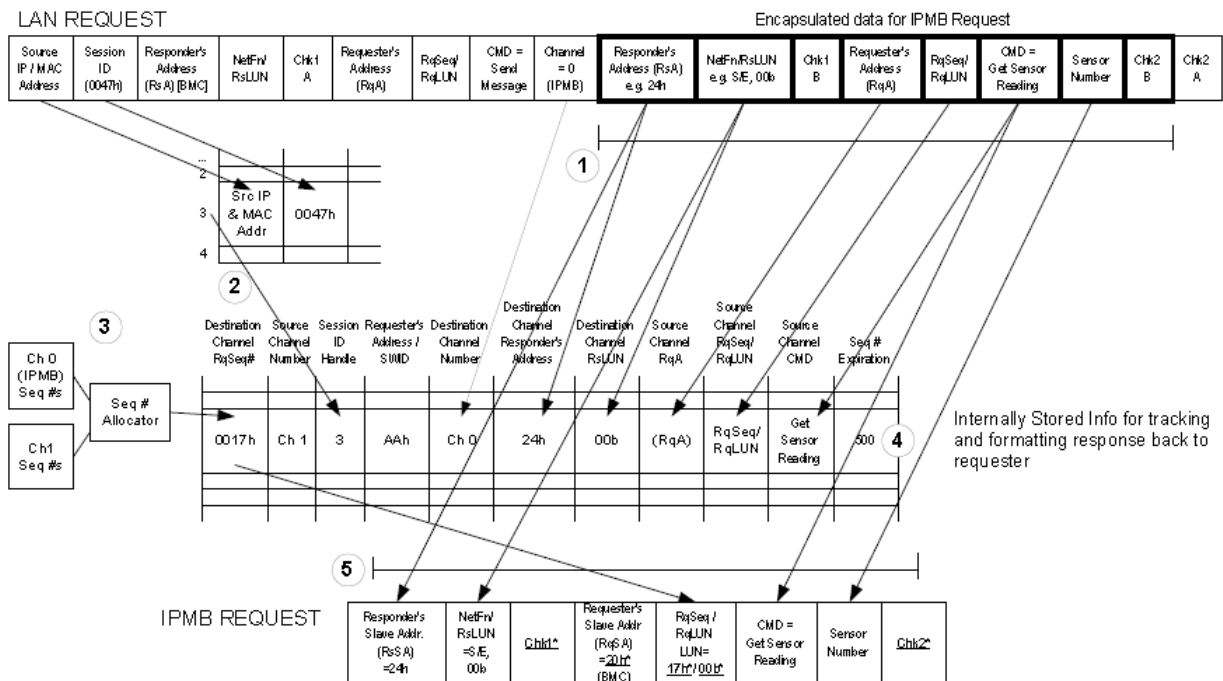
When a request message is bridged to another channel by encapsulating it in a Send Message command (from a source channel other than the system interface), the MC immediately returns a response to the Send Message command itself. Meanwhile, the request is extracted from the Send Message command and forwarded to the specified target channel.

The Send Message command must be configured to direct the MC to keep track of data in the request so when the response comes back from the target device it can be forwarded by the MC back to the channel that delivered the original Send Message command to the MC. When the response comes back from the target, the MC uses the tracking information to form at the response for the given channel. To the party that initiated the Send Message command, the response will appear as if the encapsulated request was directly executed by the MC.

For example, suppose a Get Device ID command has been encapsulated in a Send Message command directed to the IPMB from a LAN channel. The MC will immediately send a response to the Send Message command back on the LAN. The MC will extract the encapsulated Get Device ID message content and format it as a Get Device ID request for IPMB. The target device on IPMB responds with a Get Device ID response message in IPMB format. The MC takes the tracking information that was stored when the Send Message command was issued, and uses it to create a Get Device ID response in LAN format. The Responder's address information in that response can either be that of the MC, of the device on IPMB that the request was targeted at the choice of the MC implementation. Parties that initiate this type of bridged request using the Send Message command should accept responses from the MC that use either address.

The following figure and steps present an example high-level design for handling a bridged request. Note that the example shows information that is generated and stored, but it does not show any particular code module that would perform that operation. That is, the choice of which functions are centralized, which are in a LAN module, and which are in an IPMB module is left to the implementer.

Figure 3 LAN to IPMB bridged request example



1. When the MC receives the `Send Message` command with the Bridged Request parameter bit set, it checks for an available entry in a Pending Bridged Response table and copies parameters from the request to be bridged. When the response comes back, these parameters will be used to validate that the response matches the earlier request and to reformat the response for the originating channel. The bold outlined boxes represent parameters and data in the `Send Message` command that will ultimately be copied to the resulting request on the target channel.
2. Any channel session information necessary to get the response back to the original requester will also need to be recorded. In this example, the MC maintains a separate table of session information for the LAN channel. An offset into that table is used as a handle for identifying the session information associated with the request. This handle is used in the Pending Bridged Response table in place of copying all the session information. Note that with such an implementation, it is important to remember details such as invalidating and freeing any bridge table entry associated with that session if the session should get closed while responses are pending.
3. In this example, the MC has a separate Sequence Number Allocator routine that ensures that sequence numbers used in bridged requests are kept unique for a given channel. This is done so that the response comes back, the sequence number can be used to look up corresponding request info entries from the Pending Bridged Response table.
4. Responses have a five second Sequence Number Expiration interval. If a response is not received by the expiration interval, the corresponding entry in the Pending Bridged Response entry is deleted and the sequence number associated with the request can be reused. The Sequence Number Expiration column in this example represents a possible implementation where the value is decremented nominally once every 10 ms. The entry is considered to be free when the number reaches 0. In this example the Sequence Number Expiration field could be used both for tracking sequence number expiration as well as a mechanism for marking availability of the table entry.
5. The MC takes the indicated values and uses them to construct the bridged request. The request is a combination of field values copied from the original `Send Message` command and values generated by the MC. The MC generated values are shown with a bold, underlined typeface with an asterisk.

IPMB access via master write-read command

When an IPMB is implemented in the system, the MC serves as a controller to give system software access to the IPMB. The IPMB allows non-intelligent devices as well as management controllers on the bus. To support this operation, the MC provides the Master Write-Read command via its interface with system software. The Master Write-Read command provides low-level access to non-intelligent devices on the IPMB, such as FRU EEPROMs.

The Master Write-Read command provides a subset of the possible I²C and SM BUS operations that covers most I²C/SM BUS-compatible devices.

In addition to supporting non-intelligent devices on the IPMB, the Master Write-Read command also provides access to non-intelligent devices on Private Busses behind management controllers. The main purpose of this is to support FRU EEPROMs on Private Busses.

MC IPMB LUNs

A MC supports several LUNs to which messages are sent via the IPMB interface. These LUNs are used to identify different sub-addresses within the MC.

In Moonshot, since the system interface is only implemented at the system node controller, the SMS message LUN is only applicable to the system interface node controllers.

Table 121 MC IPMB LUNs

LUN	Short Description	Long Description
00b	MC commands and Event Request Messages	Event Request Messages received on this LUN are routed to the Event Receiver Function in the MC and automatically logged if SEL logging is enabled.
01b	OEM LUN 1	OEM — reserved for MC implementer/system integrator definition.
10b	SMS Message LUN (intended for messages to System Management Software)	Messages received on this LUN are routed to the Receiver Message Queue and retrieved using a Read Message command. The SMS_Avail flag is set whenever the Receive Message Queue has valid contents.
11b	OEM LUN 2	OEM — reserved for MC implementer/system integrator definition

Sending Messages to IPMB from system software

NOTE: In Moonshot, since the system interface is only implemented at the system node controller, the SMS message LUN is only applicable to the system interface node controllers.

SMS can use the MC for sending and receiving IPMB messages. Both IPMB request and response messages can be sent and received using this mechanism. Therefore, not only can system software send requests to the IPMB and receive responses from the IPMB, it is also possible for system software to receive requests from the IPMB to send back IPMB responses.

System software sends messages to the IPMB through the system interface using the MC as an IPMB controller. This is accomplished by using the `Send Message` command to write the message to the IPMB (channel 0). The MC does not place any restrictions on the type or content of the IPMB message being sent. System management software can send any IPMB request or response message it desires provided that the message meets the maximum length requirements of the `Send Message` command.

System Management Software is responsible for providing all fields for the IPMB message, including Requester and Responder Slave addresses and checksums. The following figures show an example using the `Send Message` command to send a `Set Event Receiver` command to an IPMB device at slave address 52h, LUN 00b, via the system interface. The example command sets the Event Receiver address to 20h — BMC.

The heavy bordered fields show the bytes for the IPMB message carried in the `Send Message` command. The requesters LUN field (`rqLUN`) is set to 10b (MC SMS LUN). This directs the responder to send the response to the `Set Event Receiver` command to the system node's Receive Message Queue.

Figure 4 IPMB request sent using `Send Message` command

NetFn (06h = App request)	LUN (00b)	Command (Send Message)	Channel (00h)
Slave address for write (52h = rsSA)	NetFn/rsLUN (04h / 00b = Sensor/Event, LUN 00b)		check 1 (9Eh)
rqSA (20h = BMC)	rqSeq/rqLUN (000001b / 10b, 10b = SMS LUN)		Cmd (00h = Set Event Receiver)
event receiver slave address (20h = BMC)	event receiver LUN (00h)	check 2 (BAh)	

Figure 5 `Send Message` command response

NetFn (07h = App response)	LUN (00b)	Command (Send Message)	Completion Code (00h)
-------------------------------	--------------	---------------------------	--------------------------

The response is for the `Send Message` command and not for the `Set Event Receiver` command. The response to the `Set Event Receiver` command is returned later in the `Receive Message Queue`. System software uses the `Get Message` command to read messages from the `Receive Message Queue`. System software keeps track of any outstanding responses and matches responses up with corresponding requests as they are returned. System software must also keep track of the protocol assigned to the particular channel in order to interpret the response to the `Get Message` command.

Keyboard Controller Style Interface

The Keyboard Controller Style (KCS) is one of the supported MC to SMS interfaces. The KCS interface is specified solely for SMS messages.

The KCS Interface supports polled operations. Implementations optionally provide an interrupt driven by the OBF flag, this must not prevent driver software from using the interface in a polled manner. This allows software to default to polled operation. It also allows software to use the KCS interface in a polled mode until it determines the type of interrupt support. Methods for assigning and enabling such an interrupt are outside the scope of this specification.

KCS Interface/MC LUNs

LUN 00b is typically used for all messages to the MC through the KCS Interface. LUN 10b is reserved for `Receive Message Queue` use and should not be used for sending commands to the MC. Note that messages encapsulated in a `Send Message` command can use any LUN in the encapsulated portion.

KCS Interface-MC Request message format

Request Messages are sent to the MC from system software using a write transfer through the KCS interface. The message bytes are organized according to the following format specification:

Figure 6 KCS Interface/MC Request Message format

Byte 1	Byte 2	Byte 3:N
NetFn/LUN	Cmd	Data

Where:

LUN	This is a sub-address that allows messages to be routed to different LUNS that reside behind the same physical interface. The LUN field occupies the least significant two bits of the first message byte.
NetFn	Provides the first level of functional routing for messages received by the MC via the KCS Interface. The NetFn field occupies the most significant six bits of the first message byte. Even NetFn values are used for requests to the MC, and odd NetFn values are returned in responses from the MC.
Cmd	This message byte specifies the operation that is to be executed under the specified Network Function.
Data	Zero or more bytes of data, as required by the given command. The general convention is to pass data LS-byte first, but check the individual command specifications to be sure.

MC-KCS Interface Response Message format

Response Messages are Read Transfers from the MC to system software via the KCS interface. The MC only returns responses via the KCS Interface when data needs to be returned. The message bytes are organized according to the following format specification:

Byte 1	Byte 2	Byte 3	Byte 4:N
NetFn/LUN	Cmd	Completion Code	Data

Where:

LUN	Returns the LUN that was passed in the Request Message.
NetFn	A return of the NetFn code that was passed in the Request Message. Except that an odd NetFn value is returned.
Cmd	Return of the Cmd code that was passed in the Request Message.
Completion Code	Indicates whether the request completed successfully.
Data	Zero or more bytes of data. The MC always returns a response to acknowledge the request, regardless of whether data is returned.

LAN Interface

The LAN interface specifications define how IPMI messages can be sent to and from the MC encapsulated in Remote Management Control Protocol (RMCP) packet datagrams. This capability is also referred to as IPMI over LAN. IPMI also defines the associated LAN specific configuration interfaces for settings things such as IP address other options, as well as commands for discovering IPMI based systems.

The Distributed Management Task Force (DMTF) specifies the RMCP format. This same packet format is used for non-IPMI messaging via the DMTF's Alert Standard Forum (ASF) specification. Using the RMCP packet format enables more commonality between management applications that operate in an environment that includes both IPMI based and ASF based systems.

IPMI v2.0 defines and extended packet format and capabilities that are collectively referred to as RMCP+ and defined under the IPMI specific portion of an RMCP packet. RMCP+ utilizes authentication algorithms that are more closely aligned with the mechanisms used for the ASF 2.0 specification. In addition, RMCP+ adds data confidentiality (encryption) and payload capability. In Moonshot, IPMI v2.0/RMCP+ is implemented for the LAN interface.

Remote Management Control Protocol (RMCP)

The Distributed Management Task Force (DMTF) has defined a RMCP for supporting pre-OS and OS absent management. RCMP is a simple request-response protocol that can be delivered using UDP datagrams. IPMI-over-LAN uses version 1 of the RMCP protocol and packet format.

RMCP includes a field that indicates the class of messages that can be embedded in an RMCP message packet, including a class for IPMI messages. Other message classes are ASF and OEM.

An IPMI LAN implementation can also use ASF-class Ping and Pong messages to support the discovery of IPMI managed systems on the network.

RMCP port numbers

RMCP uses two well-known ports under UDP. [“RMCP Port Numbers” \(page 154\)](#) describes these ports and summarizes their use.

Table 122 RMCP Port Numbers

Port#	Name	Description
623 (26Fh)	Aux bus Shunt (Primary RMCP Port)	The Primary RMCP Port — This port and the required RMCP messages must be provided to conform with RMCP specifications. There is a mandatory set of messages required to be supported on this port. These messages are always sent in the clear so that system software can discover systems that have RMCP support.
664 (298h)	Secure Aux BUS (Secondary RMCP Port)	Referred to as the secondary RMCP port or secure port, it is only used when it is necessary to encrypt packets using an algorithm or specification that prevents also sending unencrypted packets from being transferred via the same port.

Table 122 RMCP Port Numbers *(continued)*

Port#	Name	Description
		<p>Since discovery requires sending in the clear RMCP ping/pong packets, the secondary port is used to transfer encrypted transfers while the primary port continues to support unencrypted packets.</p> <p>An implementation that utilizes this port must still support the Primary RMCP port and the required messages on that port in order to be conformant with the RMCP specifications.</p> <p>Note that the common IPMI messaging protocols and authentication mechanisms in this specification do not use encrypted packets at the RMCP level (encrypted packets in IPMI are defined under the IPMI message class), therefore IPMI messaging does not need to use the secondary port.</p>

RMCP Message Format

There are two types of RMCP messages: Data or Normal RMCP messages and RMCP acknowledge messages. Data messages and ACK messages are differentiated by the ACK/normal bit of the class of message field.

Table 123 RMCP Message Format

Field	Size in bytes	Description
RMCP Header		
Version	1	06h = RMCP Version 1.0
Reserved	1	00h
Sequence Number	1	Varies, see text
Class of Message	1	<p>This field identifies the form of the messages that follow this header. All messages of class ASF (6) conform to the formats defined in this specification and can be extended via an OEM IANA.</p> <p>Bit 7 RMCP ACK</p> <p>0 — Normal RMCP message</p> <p>1 — RMCP ACK message</p> <p>Bit 6:5 Reserved</p> <p>Bit 4:0 Message Class</p> <p>0–5 = Reserved</p> <p>6 = ASF</p> <p>7 = IPMI</p> <p>8 = OEM defined</p> <p>all other = Reserved</p>
RMCP Data		
Data	Variable	data based class of message

The following table presents how the ACK/Normal Bit and the message class combine to identify the type of message under RMCP and which specification defines the format of the associated message data.

Table 124 Message Type Determination Under RMCP

ACK/Normal bit	Message Class	Message Type	Message Data
ACK	ASF	RMCP ACK	No Data. Message contains only RMCP heading, and sequence number from the last message received.
ACK	all other	undefined	not allowed

Table 124 Message Type Determination Under RMCP *(continued)*

ACK/Normal bit	Message Class	Message Type	Message Data
normal	ASF	ASF Messages	Per ASF specification
normal	OEM	OEM message under RMCP	bytes:3 = OEM IANA bytes 4:N = OEM message data (defined by manufacturer or organization identified by the OEM IANA field value)
normal	IPMI	IPMI messages	Per this specification

Serial Over LAN (SOL)

SOL is the name for the redirection of baseboard serial controller traffic over an IPMI session. This enables asynchronous serial-based OS and pre-OS communication over a connection to the MC. SOL can be used to provide a user at a remote console a means of interacting with serial text-based applications over the IPMI LAN session. A single remote console application can use SOL to simultaneously provide LAN access to IPMI platform management and serial text redirection under a unified user interface. SOL is implemented as a payload type under the IPMI v2.0 RMCP+ protocol. Access privileges for SOL are managed under the same user configuration interfaces that are used for IPMI management. This simplifies the creation of configuration software, remote management applications, and cross-platform configuration utilities.

7 Support and other resources

Information to collect before contacting HP

Be sure to have the following information available before you contact HP:

- Software product name
- Hardware product model number
- Operating system type and version
- Applicable error message
- Third-party hardware or software
- Technical support registration number (if applicable)

How to contact HP

Use the following methods to contact HP technical support:

- In the United States, see the Customer Service / Contact HP United States website for contact options:
<http://www.hp.com/go/assistance>
- In the United States, call +1 800 334 5144 to contact HP by telephone. This service is available 24 hours a day, 7 days a week. For continuous quality improvement, conversations might be recorded or monitored.
- In other locations, see the Contact HP Worldwide website for contact options:
<http://www.hp.com/go/assistance>

HP authorized resellers

For the name of the nearest HP authorized reseller, see the following sources:

- In the United States, see the HP U.S. service locator web site:
http://www.hp.com/service_locator
- In other locations, see the Contact HP worldwide web site:
<http://welcome.hp.com/country/us/en/wwcontact.html>

Related information

Documents

- *HP Moonshot Documentation Overview*
- *HP Moonshot Configuration and Compatibility Guide*
- *HP Moonshot 1500 Chassis Setup and Installation Guide*
- *HP Moonshot 1500 Chassis Maintenance and Service Guide*
- *Important Download Documentation, Drivers, and Software and Firmware Updates*
- *HP Moonshot Troubleshooting Guide*
- *Safety, Compliance, and Warranty Information*
- *HP Moonshot iLO Chassis Management Firmware User Guide*
- *HP Moonshot Component Pack Release Notes*

These documents are on the HP website at:

<http://www.hp.com/go/moonshot/docs>

Websites

- HP Moonshot website:
<http://www.hp.com/go/moonshot>
- HP Moonshot Component Pack download website:
<http://www.hp.com/go/servers/moonshot/download>
- Intel IPMI specification website:
<http://www.intel.com/design/servers/ipmi/tools.htm>

A Command Assignments

The following lists the commands defined in this specification and the minimum privilege level required to execute a given command. In addition, the following apply:

- Unless otherwise specified, unauthenticated, session-less interfaces, such as the System Interface and IPMB, can support any IPMI command.
- The privilege level requirements for OEM commands (NetFn=OEM, OEM/Group) is specified by the OEM identified by the corresponding manufacturer ID.
- Note that the `Send Message` and `Master Write-Read` commands are not available at the User privilege level, with the exception of using a `Send Message` command to deliver a message to the System Interface. This is because these commands enable unfiltered access the IPMB, ICMB, private management busses, and PCI Management Bus. This would potentially allow someone to use those commands to send commands to other controllers or write to non-intelligent devices on those busses. As a consequence, a User is only able to read FRU and sensors directly managed by the MC. In addition, FRU must be accessed via the `Read FRU` command and not `Master Write-Read`.
- The `Send Message` command can be used to deliver a message to the System Interface at User privilege level. It is up to the system software to determine the privilege level and place any additional restrictions on messages received via the `Receive Message Queue`. This can be accomplished by using the session handle associated with the message and the `Get Session Info` command to look up the privilege level that the user is operating at. Software can also check the limits for the channel and the user by using information from the `Get Channel Access` and `Get User Access` commands to determine whether a given user has sufficient privilege to deliver a particular command to system software.

Unless otherwise specified, the listed IPMI commands, if supported, must be accessible via LUN 00b.

Key for Table 125 (page 160)

- b = Command only generated by MC, can be sent prior to a session being established
- b1 = Command only generated by MC, can only be delivered to a session-less channel, or a channel that has an active session
- b2 = Command only generated by MC, can be sent to a serial channel when serial port sharing is used and activating the SOL payload causes the serial session to be terminated.
- b3 = Command only generated by MC, can only be delivered to a session-less channel.
- p = Works at any privilege level, can be sent prior to a session being established
- s = Command executable via system interface only
- X = Supported at given privilege level or higher
- I = Command executable from local interfaces only (e.g. IPMB, SMBus, PCI Mgmt. bus or System Interface)
- C = Callback privilege
- U = User Privilege level
- O = Operator Privilege level
- A = Administrator Privilege level
- App = Application Network Function Code
- S/E = Sensor/Event Network Function Code
- - = Reserved/unassigned, or OEM specified

Table 125 Moonshot command number assignments and privilege levels

	NetFn	CMD	C	U	O	A
IPM Device “Global” Commands						
Get Device ID	App	01h		X		
Broadcast ‘Get Device ID’ ¹	App	01h	I	I	I	I
Cold Reset	App	02h				X
Warm Reset	App	03h				X
Get Self Test Results	App	04h		X		
Get ACPI Power State	App	07h		X		
MC Watchdog Timer Commands						
Reset Watchdog Timer	App	22h			X	
Set Watchdog Timer	App	24h			X	
Get Watchdog Timer	App	25h		X		
MC Device and Messaging Commands						
Set BMC Global Enables	App	2Eh	s	s	s	s
Get BMC Global Enables	App	2Fh		X		
Clear Message Flags	App	30h	s	s	s	s
Get Message Flags	App	31h	s	s	s	s
Enable Message Channel Receive	App	32h	s	s	s	s
Get Message	App	33h	s	s	s	s
Send Message	App	34h		X ²	X	
Get System GUID	App	37h	p ³	p ³	p ³	p ³
Set System Info Parameters	App	58h				X
Get System Info Parameters	App	59h		X		
Set Session Privilege Level	App	3Bh		X ⁴		
Close Session	App	3Ch	X ⁵			
Get Session Info	App	3Dh		X		
Get AuthCode	App	3Fh			X	
Set Channel Access	App	40h				X
Get Channel Access	App	41h		X		
Get Channel Info	App	42h		X		
Set User Access	App	43h				X
Get User Access	App	44h			X	
Set User Name	App	45h				X
Get User Name	App	46h			X	
Set User Password	App	47h				X
Activate Payload	App	48h	X ⁵	U	U	U
Deactivate Payload	App	49h	X ⁵	U	U	U

Table 125 Moonshot command number assignments and privilege levels *(continued)*

	NetFn	CMD	C	U	O	A
Get Payload Activation Status	App	4Ah		X		
Get Payload Instance Info	App	4Bh		X		
Set User Payload Access	App	4Ch				X
Get User Payload Access	App	4Dh			X	
Get Channel Payload Support	App	4Eh		X		
Get Channel Payload Version	App	4Fh		X		
Master Write-Read	App	52h			X	
Get Channel Cipher Suites	App	54h	p	p	p	p
Suspend/Resume Payload Encryption	App	55h		X ⁹		
Set Channel Security Keys	App	56h				X
Get System Interface Capabilities	App	57h		X		
Chassis Device Commands						
Get Chassis Capabilities	Chassis	00h		X		
Get Chassis Status	Chassis	01h		X		
Chassis Control	Chassis	02h			X	
Chassis Identify	Chassis	04h			X	
Set Power Restore Policy	Chassis	06h			X	
Set System Boot Options	Chassis	08h			X ⁶	
Get System Boot Options	Chassis	09h			X	
unassigned	Chassis	0A-0Ch	-	-	-	-
Get POH Counter	Chassis	0Fh		X		
Event Commands						
Set Event Receiver	S/E	00h				X
Get Event Receiver	S/E	01h		X		
Platform Event (a.k.a. "Event Message")	S/E	02h			X	
unassigned	S/E	03h-0Fh	-	-	-	-
Sensor Device Commands						
Get Device SDR Info	S/E	20h	I	I	I	I
Get Device SDR	S/E	21h	I	I	I	I
Reserve Device SDR Repository						
Get Sensor Reading	S/E	2Dh		X		
FRU Device Commands						
Get FRU Inventory Area Info	Storage	10h		X		
Read FRU Data	Storage	11h		X		

Table 125 Moonshot command number assignments and privilege levels *(continued)*

	NetFn	CMD	C	U	O	A
Write FRU Data	Storage	12h			X	
SDR Device Commands						
Get SDR Repository Info	Storage	20h		X		
Get SDR Repository Allocation Info	Storage	21h		X		
Reserve SDR Repository	Storage	22h		X		
Get SDR	Storage	23h		X		
Add SDR	Storage	24h			X	
Delete SDR	Storage	26h			X	
Clear SDR Repository	Storage	27h			X	
Run Initialization Agent	Storage	2Ch			X	
SEL Device Commands						
Get SEL Info	Storage	40h		X		
Reserve SEL	Storage	42h		X		
Get SEL Entry	Storage	43h		X		
Add SEL Entry	Storage	44h			X	
Clear SEL	Storage	47h			X	
Get SEL Time	Storage	48h		X		
Set SEL Time	Storage	49h			X	
LAN Device Commands						
Set LAN Configuration Parameters	Transport	01h				X
Get LAN Configuration Parameters	Transport	02h			X	
Serial/Modem Device Commands						
Set SOL Configuration Parameters	Transport	21h				X
Get SOL Configuration Parameters	Transport	22h		X		
DCMI Specific						
Get DCMI Capability Info	DCGRP (2ch)	01h		X		
Get Asset Tag	DCGRP (2ch)	06h		X		
Get DCMI Sensor Info	DCGRP (2ch)	07h			X	
Set Asset Tag	DCGRP (2ch)	08h			X	
Get Controller ID String	DCGRP (2ch)	09h		X		
Set Controller ID String	DCGRP (2ch)	0Ah			X	
PICMG Specific						

Table 125 Moonshot command number assignments and privilege levels *(continued)*

	NetFn	CMD	C	U	O	A
Get PICMG Properties	PICMG (00h)	00h		X		
Get Address Info	PICMG (00h)	01h		X		
FRU Inventory Device Lock Control	PICMG (00h)	1Fh			X	

¹ This command is sent using the Broadcast format on IPMB. See command description for details.

² A User can use a `Send Message` command to deliver a message to system software, but Operator privilege is required to use it to access other channels.

³ Command only applies to authenticated channels.

⁴ This is effectively a no-op if the user has a maximum privilege limit of User since the command could not be used to change the operating privilege level to a higher value.

⁵ A session operating at Callback, User, or Operator can only use this command to terminate their own session. An Administrator or system software can use the command to terminate any session.

⁶ There is a bit in this command that can only be set at Administrator privilege level

⁷ Command available for all levels except for User level.

⁸ See [ICMB] specification for command specifications

⁹ The Suspend/Resume Payload Encryption command may be overridden by a configuration option for the particular payload type that forces encryption to be used. In this case, an Admin level command would typically be required to change the configuration.

¹⁰ The configuration parameters for a given payload type determine the privilege level required to activate/deactivate the payload.

B Verbose output examples

```
root@JSMITH-LX:/# ipmitool -I lanplus -H ILOH101GEMINI -U Administrator -P password sdr list all -v
```

```
Running Get PICMG Properties my_addr 0x20, transit 0, target 0x20
Discovered PICMG Extension 2.3
Discovered IPMB-0 address 0x20
Device ID : ZoMC
Entity ID : 240.96 (PICMG Shelf Management Controller)
Device Slave Address : 20h
Channel Number : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence : Static
Logs Init Agent Errors : No
Event Message Gen : Enable
Device Capabilities
Chassis Device : No
Bridge : No
IPMB Event Generator : No
IPMB Event Receiver : Yes
FRU Inventory Device : Yes
SEL Device : Yes
SDR Repository : Yes
Sensor Device : Yes

Device ID : 254
Entity ID : 240.96 (PICMG Shelf Management Controller)
Device Access Address : 20h
Logical FRU Device : FEh
Channel Number : 0h
LUN.Bus : 0h.0h
Device Type.Modifier : 10h.0h (IPMI FRU Inventory)
OEM : 00h

Sensor ID : IPMB0 Phys Link (0x1)
Entity ID : 240.96 (PICMG Shelf Management Controller)
Sensor Type (Discrete): PICMG IPMB0 Link State (0xf1)
Sensor Reading : 0h
Event Message Control : No Events From Sensor
OEM : 0

Device ID : ChasMgmtCtrl1
Entity ID : 23.1 (System Chassis)
Device Slave Address : 44h
Channel Number : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence : Static
Logs Init Agent Errors : No
Event Message Gen : Enable
Device Capabilities
Chassis Device : Yes
Bridge : No
IPMB Event Generator : Yes
IPMB Event Receiver : No
FRU Inventory Device : Yes
SEL Device : No
SDR Repository : Yes
Sensor Device : Yes

Device ID : PsMgmtCtrl1
Entity ID : 10.1 (Power Supply)
Device Slave Address : 52h
Channel Number : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence : Dynamic
Logs Init Agent Errors : No
Event Message Gen : Enable
Device Capabilities
Chassis Device : No
Bridge : No
IPMB Event Generator : Yes
IPMB Event Receiver : No
FRU Inventory Device : Yes
SEL Device : No
SDR Repository : Yes
Sensor Device : Yes

Device ID : PsMgmtCtrl2
Entity ID : 10.2 (Power Supply)
Device Slave Address : 54h
Channel Number : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence : Dynamic
Logs Init Agent Errors : No
Event Message Gen : Enable
```

```

Device Capabilities
Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID            : PsMgmtCtrlr3
Entity ID            : 10.3 (Power Supply)
Device Slave Address : 56h
Channel Number       : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence   : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities
Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID            : PsMgmtCtrlr4
Entity ID            : 10.4 (Power Supply)
Device Slave Address : 58h
Channel Number       : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence   : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities
Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID            : CaMC
Entity ID            : 160.97 (PICMG Front Board)
Device Slave Address : A6h
Channel Number       : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence   : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities
Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID            : CaMC
Entity ID            : 160.97 (PICMG Front Board)
Device Slave Address : A8h
Channel Number       : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence   : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities
Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID            : CaMC
Entity ID            : 160.97 (PICMG Front Board)
Device Slave Address : AAh
Channel Number       : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence   : Dynamic

```

```

Logs Init Agent Errors : No
Event Message Gen      : Enable
Device Capabilities
Chassis Device         : No
Bridge                 : No
IPMB Event Generator   : Yes
IPMB Event Receiver    : No
FRU Inventory Device   : Yes
SEL Device             : No
SDR Repository         : Yes
Sensor Device          : Yes

Device ID              : CaMC
Entity ID              : 160.97 (PICMG Front Board)
Device Slave Address   : ACh
Channel Number         : 0h
ACPI System P/S Notif  : Not Required
ACPI Device P/S Notif  : Not Required
Controller Presence    : Dynamic
Logs Init Agent Errors : No
Event Message Gen      : Enable
Device Capabilities
Chassis Device         : No
Bridge                 : No
IPMB Event Generator   : Yes
IPMB Event Receiver    : No
FRU Inventory Device   : Yes
SEL Device             : No
SDR Repository         : Yes
Sensor Device          : Yes

Device ID              : CaMC
Entity ID              : 160.97 (PICMG Front Board)
Device Slave Address   : AEh
Channel Number         : 0h
ACPI System P/S Notif  : Not Required
ACPI Device P/S Notif  : Not Required
Controller Presence    : Dynamic
Logs Init Agent Errors : No
Event Message Gen      : Enable
Device Capabilities
Chassis Device         : No
Bridge                 : No
IPMB Event Generator   : Yes
IPMB Event Receiver    : No
FRU Inventory Device   : Yes
SEL Device             : No
SDR Repository         : Yes
Sensor Device          : Yes

Device ID              : CaMC
Entity ID              : 160.97 (PICMG Front Board)
Device Slave Address   : B0h
Channel Number         : 0h
ACPI System P/S Notif  : Not Required
ACPI Device P/S Notif  : Not Required
Controller Presence    : Dynamic
Logs Init Agent Errors : No
Event Message Gen      : Enable
Device Capabilities
Chassis Device         : No
Bridge                 : No
IPMB Event Generator   : Yes
IPMB Event Receiver    : No
FRU Inventory Device   : Yes
SEL Device             : No
SDR Repository         : Yes
Sensor Device          : Yes

Device ID              : CaMC
Entity ID              : 160.97 (PICMG Front Board)
Device Slave Address   : B2h
Channel Number         : 0h
ACPI System P/S Notif  : Not Required
ACPI Device P/S Notif  : Not Required
Controller Presence    : Dynamic
Logs Init Agent Errors : No
Event Message Gen      : Enable
Device Capabilities
Chassis Device         : No
Bridge                 : No
IPMB Event Generator   : Yes
IPMB Event Receiver    : No
FRU Inventory Device   : Yes
SEL Device             : No
SDR Repository         : Yes
Sensor Device          : Yes

Device ID              : CaMC
Entity ID              : 160.97 (PICMG Front Board)
Device Slave Address   : B8h
Channel Number         : 0h
ACPI System P/S Notif  : Not Required

```

```

ACPI Device P/S Notif : Not Required
Controller Presence   : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities
Chassis Device        : No
Bridge                : No
IPMB Event Generator  : Yes
IPMB Event Receiver   : No
FRU Inventory Device  : Yes
SEL Device            : No
SDR Repository        : Yes
Sensor Device         : Yes

Device ID              : CaMC
Entity ID              : 160.97 (PICMG Front Board)
Device Slave Address   : B4h
Channel Number        : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence    : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities
Chassis Device        : No
Bridge                : No
IPMB Event Generator  : Yes
IPMB Event Receiver   : No
FRU Inventory Device  : Yes
SEL Device            : No
SDR Repository        : Yes
Sensor Device         : Yes

Device ID              : CaMC
Entity ID              : 160.97 (PICMG Front Board)
Device Slave Address   : BAh
Channel Number        : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence    : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities
Chassis Device        : No
Bridge                : No
IPMB Event Generator  : Yes
IPMB Event Receiver   : No
FRU Inventory Device  : Yes
SEL Device            : No
SDR Repository        : Yes
Sensor Device         : Yes

Device ID              : CaMC
Entity ID              : 160.97 (PICMG Front Board)
Device Slave Address   : 82h
Channel Number        : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence    : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities
Chassis Device        : No
Bridge                : No
IPMB Event Generator  : Yes
IPMB Event Receiver   : No
FRU Inventory Device  : Yes
SEL Device            : No
SDR Repository        : Yes
Sensor Device         : Yes

Device ID              : CaMC
Entity ID              : 160.97 (PICMG Front Board)
Device Slave Address   : B6h
Channel Number        : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence    : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities
Chassis Device        : No
Bridge                : No
IPMB Event Generator  : Yes
IPMB Event Receiver   : No
FRU Inventory Device  : Yes
SEL Device            : No
SDR Repository        : Yes
Sensor Device         : Yes

Device ID              : CaMC
Entity ID              : 160.97 (PICMG Front Board)
Device Slave Address   : BCh

```

```

Channel Number      : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities
Chassis Device       : No
Bridge               : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID           : CaMC
Entity ID           : 160.97 (PICMG Front Board)
Device Slave Address : 84h
Channel Number      : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities
Chassis Device       : No
Bridge               : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID           : CaMC
Entity ID           : 160.97 (PICMG Front Board)
Device Slave Address : BEh
Channel Number      : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities
Chassis Device       : No
Bridge               : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID           : CaMC
Entity ID           : 160.97 (PICMG Front Board)
Device Slave Address : 86h
Channel Number      : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities
Chassis Device       : No
Bridge               : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID           : CaMC
Entity ID           : 160.97 (PICMG Front Board)
Device Slave Address : C0h
Channel Number      : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities
Chassis Device       : No
Bridge               : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID           : CaMC

```



```

Entity ID           : 160.97 (PICMG Front Board)
Device Slave Address : 88h
Channel Number      : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen    : Enable
Device Capabilities
Chassis Device       : No
Bridge               : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID           : CaMC
Entity ID           : 160.97 (PICMG Front Board)
Device Slave Address : C2h
Channel Number      : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen    : Enable
Device Capabilities
Chassis Device       : No
Bridge               : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID           : CaMC
Entity ID           : 160.97 (PICMG Front Board)
Device Slave Address : C4h
Channel Number      : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen    : Enable
Device Capabilities
Chassis Device       : No
Bridge               : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID           : CaMC
Entity ID           : 160.97 (PICMG Front Board)
Device Slave Address : 8Ah
Channel Number      : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen    : Enable
Device Capabilities
Chassis Device       : No
Bridge               : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID           : CaMC
Entity ID           : 160.97 (PICMG Front Board)
Device Slave Address : C6h
Channel Number      : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen    : Enable
Device Capabilities
Chassis Device       : No
Bridge               : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

```

```

Device ID          : CaMC
Entity ID          : 160.97 (PICMG Front Board)
Device Slave Address : 8Ch
Channel Number     : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence : Dynamic
Logs Init Agent Errors : No
Event Message Gen   : Enable
Device Capabilities
Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver : No
FRU Inventory Device : Yes
SEL Device          : No
SDR Repository      : Yes
Sensor Device       : Yes

```

```

Device ID          : CaMC
Entity ID          : 160.97 (PICMG Front Board)
Device Slave Address : C8h
Channel Number     : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence : Dynamic
Logs Init Agent Errors : No
Event Message Gen   : Enable
Device Capabilities
Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver : No
FRU Inventory Device : Yes
SEL Device          : No
SDR Repository      : Yes
Sensor Device       : Yes

```

```

Device ID          : CaMC
Entity ID          : 160.97 (PICMG Front Board)
Device Slave Address : 8Eh
Channel Number     : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence : Dynamic
Logs Init Agent Errors : No
Event Message Gen   : Enable
Device Capabilities
Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver : No
FRU Inventory Device : Yes
SEL Device          : No
SDR Repository      : Yes
Sensor Device       : Yes

```

```

Device ID          : CaMC
Entity ID          : 160.97 (PICMG Front Board)
Device Slave Address : 90h
Channel Number     : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence : Dynamic
Logs Init Agent Errors : No
Event Message Gen   : Enable
Device Capabilities
Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver : No
FRU Inventory Device : Yes
SEL Device          : No
SDR Repository      : Yes
Sensor Device       : Yes

```

```

Device ID          : CaMC
Entity ID          : 160.97 (PICMG Front Board)
Device Slave Address : CAh
Channel Number     : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence : Dynamic
Logs Init Agent Errors : No
Event Message Gen   : Enable
Device Capabilities
Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver : No
FRU Inventory Device : Yes
SEL Device          : No

```

```

SDR Repository      : Yes
Sensor Device       : Yes

Device ID           : CaMC
Entity ID           : 160.97 (PICMG Front Board)
Device Slave Address : CCh
Channel Number      : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen    : Enable
Device Capabilities

Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device          : No
SDR Repository      : Yes
Sensor Device       : Yes

Device ID           : CaMC
Entity ID           : 160.97 (PICMG Front Board)
Device Slave Address : 92h
Channel Number      : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen    : Enable
Device Capabilities

Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device          : No
SDR Repository      : Yes
Sensor Device       : Yes

Device ID           : CaMC
Entity ID           : 160.97 (PICMG Front Board)
Device Slave Address : CEh
Channel Number      : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen    : Enable
Device Capabilities

Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device          : No
SDR Repository      : Yes
Sensor Device       : Yes

Device ID           : CaMC
Entity ID           : 160.97 (PICMG Front Board)
Device Slave Address : 94h
Channel Number      : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen    : Enable
Device Capabilities

Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device          : No
SDR Repository      : Yes
Sensor Device       : Yes

Device ID           : CaMC
Entity ID           : 160.97 (PICMG Front Board)
Device Slave Address : D0h
Channel Number      : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen    : Enable
Device Capabilities

Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No

```

```

FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID            : CaMC
Entity ID            : 160.97 (PICMG Front Board)
Device Slave Address : 96h
Channel Number       : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence   : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities

Chassis Device       : No
Bridge               : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID            : CaMC
Entity ID            : 160.97 (PICMG Front Board)
Device Slave Address : D2h
Channel Number       : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence   : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities

Chassis Device       : No
Bridge               : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID            : CaMC
Entity ID            : 160.97 (PICMG Front Board)
Device Slave Address : 98h
Channel Number       : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence   : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities

Chassis Device       : No
Bridge               : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID            : CaMC
Entity ID            : 160.97 (PICMG Front Board)
Device Slave Address : D4h
Channel Number       : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence   : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities

Chassis Device       : No
Bridge               : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID            : CaMC
Entity ID            : 160.97 (PICMG Front Board)
Device Slave Address : 9Ah
Channel Number       : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence   : Dynamic
Logs Init Agent Errors : No
Event Message Gen     : Enable
Device Capabilities

Chassis Device       : No
Bridge               : No

```

```

IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device  : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID            : CaMC
Entity ID            : 160.97 (PICMG Front Board)
Device Slave Address : D6h
Channel Number       : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen    : Enable
Device Capabilities

Chassis Device       : No
Bridge               : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device  : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID            : CaMC
Entity ID            : 160.97 (PICMG Front Board)
Device Slave Address : 9Ch
Channel Number       : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen    : Enable
Device Capabilities

Chassis Device       : No
Bridge               : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device  : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID            : CaMC
Entity ID            : 160.97 (PICMG Front Board)
Device Slave Address : 9Eh
Channel Number       : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen    : Enable
Device Capabilities

Chassis Device       : No
Bridge               : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device  : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID            : CaMC
Entity ID            : 160.97 (PICMG Front Board)
Device Slave Address : D8h
Channel Number       : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen    : Enable
Device Capabilities

Chassis Device       : No
Bridge               : No
IPMB Event Generator : Yes
IPMB Event Receiver  : No
FRU Inventory Device  : Yes
SEL Device           : No
SDR Repository       : Yes
Sensor Device        : Yes

Device ID            : CaMC
Entity ID            : 160.97 (PICMG Front Board)
Device Slave Address : A0h
Channel Number       : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen    : Enable
Device Capabilities

```

```

Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver : No
FRU Inventory Device : Yes
SEL Device          : No
SDR Repository      : Yes
Sensor Device       : Yes

Device ID           : CaMC
Entity ID           : 160.97 (PICMG Front Board)
Device Slave Address : DAh
Channel Number      : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen    : Enable
Device Capabilities

Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver : No
FRU Inventory Device : Yes
SEL Device          : No
SDR Repository      : Yes
Sensor Device       : Yes

Device ID           : CaMC
Entity ID           : 160.97 (PICMG Front Board)
Device Slave Address : A2h
Channel Number      : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen    : Enable
Device Capabilities

Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver : No
FRU Inventory Device : Yes
SEL Device          : No
SDR Repository      : Yes
Sensor Device       : Yes

Device ID           : CaMC
Entity ID           : 160.97 (PICMG Front Board)
Device Slave Address : A4h
Channel Number      : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence  : Dynamic
Logs Init Agent Errors : No
Event Message Gen    : Enable
Device Capabilities

Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver : No
FRU Inventory Device : Yes
SEL Device          : No
SDR Repository      : Yes
Sensor Device       : Yes

```

At cartridge:

```
root@JSMITH-LX:/# ipmitool -I lanplus -H ILOH101GEMINI -U Administrator -P password -t 0xa4 sdr list all -v
```

```
Running Get PICMG Properties my_addr 0x20, transit 0, target 0x20
```

```

Discovered PICMG Extension 2.3
Discovered IPMB-0 address 0x20
Discovered Target IPMB-0 address 0xa4
Sensor ID           : 01-Front Ambient (0x1)
Entity ID           : 64.97 (Air Inlet)
Sensor Type (Threshold) : Temperature (0x01)
Sensor Reading      : 21 (+/- 0) degrees C
Status              : ok
Positive Hysteresis  : Unspecified
Negative Hysteresis  : Unspecified
Minimum sensor range : -127.000
Maximum sensor range : Unspecified
Event Message Control : Entire Sensor Only
Readable Thresholds  : ucr unr
Settable Thresholds  :
Threshold Read Mask  : ucr unr

Sensor ID           : 02-CPU (0x2)
Entity ID           : 65.97 (Processor)
Sensor Type (Threshold) : Temperature (0x01)
Sensor Reading      : 40 (+/- 0) degrees C
Status              : ok

```

```

Positive Hysteresis : Unspecified
Negative Hysteresis : Unspecified
Minimum sensor range : -127.000
Maximum sensor range : Unspecified
Event Message Control : Entire Sensor Only
Readable Thresholds : ucr unr
Settable Thresholds :
Threshold Read Mask : ucr unr

Sensor ID : 03-DIMM (0x3)
Entity ID : 32.97 (Memory Device)
Sensor Type (Threshold) : Temperature (0x01)
Sensor Reading : 26 (+/- 0) degrees C
Status : ok
Positive Hysteresis : Unspecified
Negative Hysteresis : Unspecified
Minimum sensor range : -127.000
Maximum sensor range : Unspecified
Event Message Control : Entire Sensor Only
Readable Thresholds : ucr unr
Settable Thresholds :
Threshold Read Mask : ucr unr

Sensor ID : 04-Cart Ctrlr (0x4)
Entity ID : 66.97 (Baseboard/Main System Board)
Sensor Type (Threshold) : Temperature (0x01)
Sensor Reading : 24 (+/- 0) degrees C
Status : ok
Positive Hysteresis : Unspecified
Negative Hysteresis : Unspecified
Minimum sensor range : -127.000
Maximum sensor range : Unspecified
Event Message Control : Entire Sensor Only
Readable Thresholds : ucr unr
Settable Thresholds :
Threshold Read Mask : ucr unr

Sensor ID : 05-CPU Zone (0x5)
Entity ID : 66.98 (Baseboard/Main System Board)
Sensor Type (Threshold) : Temperature (0x01)
Sensor Reading : 29 (+/- 0) degrees C
Status : ok
Positive Hysteresis : Unspecified
Negative Hysteresis : Unspecified
Minimum sensor range : -127.000
Maximum sensor range : Unspecified
Event Message Control : Entire Sensor Only
Readable Thresholds : ucr unr
Settable Thresholds :
Threshold Read Mask : ucr unr

Sensor ID : 06-LOM Zone (0x6)
Entity ID : 66.99 (Baseboard/Main System Board)
Sensor Type (Threshold) : Temperature (0x01)
Sensor Reading : 36 (+/- 0) degrees C
Status : ok
Positive Hysteresis : Unspecified
Negative Hysteresis : Unspecified
Minimum sensor range : -127.000
Maximum sensor range : Unspecified
Event Message Control : Entire Sensor Only
Readable Thresholds : ucr unr
Settable Thresholds :
Threshold Read Mask : ucr unr

Device ID : CaMC
Entity ID : 160.97 (PICMG Front Board)
Device Slave Address : A4h
Channel Number : 0h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence : Dynamic
Logs Init Agent Errors : No
Event Message Gen : Enable
Device Capabilities
Chassis Device : No
Bridge : No
IPMB Event Generator : Yes
IPMB Event Receiver : No
FRU Inventory Device : Yes
SEL Device : No
SDR Repository : Yes
Sensor Device : Yes

Device ID : SnMC
Entity ID : 193.97 (PICMG AdvancedMC Module)
Device Slave Address : 72h
Channel Number : 7h
ACPI System P/S Notif : Not Required
ACPI Device P/S Notif : Not Required
Controller Presence : Dynamic
Logs Init Agent Errors : No
Event Message Gen : Enable

```

```

Device Capabilities
Chassis Device      : No
Bridge              : No
IPMB Event Generator : Yes
IPMB Event Receiver : No
FRU Inventory Device : Yes
SEL Device          : Yes
SDR Repository      : Yes
Sensor Device       : Yes

Device ID           : SnMC 1
Entity ID           : 193.98 (PICMG AdvancedMC Module)
Device Access Address : A4h
Logical FRU Device   : 01h
Channel Number       : 0h
LUN.Bus             : 0h.0h
Device Type.Modifier : 10h.0h (IPMI FRU Inventory)
OEM                  : 00h

root@JSMITH-LX:/# ipmitool -I lanplus -H ILOH101GEMINI -U Administrator -P password sel list -v

Running Get PICMG Properties my_addr 0x20, transit 0, target 0x20
Discovered PICMG Extension 2.3
Discovered IPMB-0 address 0x20
SEL Record ID       : 0000
Record Type          : 02
Timestamp            : 02/23/2000 21:29:38
Generator ID         : 0044
EvM Revision         : 04
Sensor Type          : Power Supply
Sensor Number        : 04
Event Type           : Sensor-specific Discrete
Event Direction      : Assertion Event
Event Data           : 01ffff
Description           : Failure detected

SEL Record ID       : 0001
Record Type          : 02
Timestamp            : 02/28/2000 21:57:58
Generator ID         : 0044
EvM Revision         : 04
Sensor Type          : Power Supply
Sensor Number        : 04
Event Type           : Sensor-specific Discrete
Event Direction      : Assertion Event
Event Data           : 01ffff
Description           : Failure detected

SEL Record ID       : 0002
Record Type          : 02
Timestamp            : 03/06/2000 11:11:14
Generator ID         : 0044
EvM Revision         : 04
Sensor Type          : Power Supply
Sensor Number        : 04
Event Type           : Sensor-specific Discrete
Event Direction      : Assertion Event
Event Data           : 01ffff
Description           : Failure detected

SEL Record ID       : 0003
Record Type          : 02
Timestamp            : 03/17/2000 01:52:47
Generator ID         : 0044
EvM Revision         : 04
Sensor Type          : Power Supply
Sensor Number        : 04
Event Type           : Sensor-specific Discrete
Event Direction      : Assertion Event
Event Data           : 01ffff
Description           : Failure detected

SEL Record ID       : 0004
Record Type          : 02
Timestamp            : 03/24/2000 03:07:30
Generator ID         : 0044
EvM Revision         : 04
Sensor Type          : Power Supply
Sensor Number        : 04
Event Type           : Sensor-specific Discrete
Event Direction      : Assertion Event
Event Data           : 01ffff
Description           : Failure detected

```



```

SEL Record ID      : 0005
Record Type        : 02
Timestamp          : 03/27/2000 01:23:36
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 04
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event
Event Data         : 01ffff
Description        : Failure detected

SEL Record ID      : 0006
Record Type        : 02
Timestamp          : 03/27/2000 03:00:34
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 04
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event
Event Data         : 01ffff
Description        : Failure detected

SEL Record ID      : 0007
Record Type        : 02
Timestamp          : 03/27/2000 08:42:18
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 02
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event
Event Data         : 01ffff
Description        : Failure detected

SEL Record ID      : 0008
Record Type        : 02
Timestamp          : 03/27/2000 08:42:18
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 03
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event
Event Data         : 01ffff
Description        : Failure detected

SEL Record ID      : 0009
Record Type        : 02
Timestamp          : 03/27/2000 08:42:18
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 04
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event
Event Data         : 01ffff
Description        : Failure detected

SEL Record ID      : 000a
Record Type        : 02
Timestamp          : 03/27/2000 20:58:04
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 04
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event
Event Data         : 01ffff
Description        : Failure detected

SEL Record ID      : 000b
Record Type        : 02
Timestamp          : 04/04/2000 22:19:34
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 02
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event

```

```

Event Data          : 01ffff
Description         : Failure detected

SEL Record ID      : 000c
Record Type        : 02
Timestamp          : 04/04/2000 22:19:36
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 03
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event
Event Data         : 01ffff
Description        : Failure detected

SEL Record ID      : 000d
Record Type        : 02
Timestamp          : 04/04/2000 22:19:36
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 04
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event
Event Data         : 01ffff
Description        : Failure detected

SEL Record ID      : 000e
Record Type        : 02
Timestamp          : 04/04/2000 22:19:36
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 05
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event
Event Data         : 01ffff
Description        : Failure detected

SEL Record ID      : 000f
Record Type        : 02
Timestamp          : 04/07/2000 01:33:48
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 04
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event
Event Data         : 01ffff
Description        : Failure detected

SEL Record ID      : 0010
Record Type        : 02
Timestamp          : 04/07/2000 01:37:33
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 04
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event
Event Data         : 01ffff
Description        : Failure detected

SEL Record ID      : 0011
Record Type        : 02
Timestamp          : 04/16/2013 20:22:01
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 04
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event
Event Data         : 01ffff
Description        : Failure detected

SEL Record ID      : 0012
Record Type        : 02
Timestamp          : 04/16/2013 21:23:09
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply

```

```

Sensor Number      : 04
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event
Event Data         : 01ffff
Description        : Failure detected

SEL Record ID     : 0013
Record Type        : 02
Timestamp          : 04/18/2013 13:24:19
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 04
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event
Event Data         : 01ffff
Description        : Failure detected

SEL Record ID     : 0014
Record Type        : 02
Timestamp          : 04/18/2013 13:36:11
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 02
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event
Event Data         : 01ffff
Description        : Failure detected

SEL Record ID     : 0015
Record Type        : 02
Timestamp          : 04/22/2013 15:54:30
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 04
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event
Event Data         : 01ffff
Description        : Failure detected

SEL Record ID     : 0016
Record Type        : 02
Timestamp          : 06/26/2013 21:57:07
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 04
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event
Event Data         : 01ffff
Description        : Failure detected

SEL Record ID     : 0017
Record Type        : 02
Timestamp          : 06/27/2013 18:17:54
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 04
Event Type         : Sensor-specific Discrete
Event Direction    : Assertion Event
Event Data         : 01ffff
Description        : Failure detected

SEL Record ID     : 0018
Record Type        : 02
Timestamp          : 06/28/2013 20:36:17
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Power Supply
Sensor Number      : 02
Event Type         : Sensor-specific Discrete
Event Direction    : Deassertion Event
Event Data         : 00ffff
Description        : Presence detected

SEL Record ID     : 0019
Record Type        : 02
Timestamp          : 07/28/2013 00:20:52

```

```

Generator ID      : 0044
EvM Revision     : 04
Sensor Type      : Power Supply
Sensor Number    : 02
Event Type       : Sensor-specific Discrete
Event Direction  : Assertion Event
Event Data       : 01ffff
Description      : Failure detected

SEL Record ID    : 001a
Record Type      : 02
Timestamp        : 08/04/2013 00:23:10
Generator ID     : 0044
EvM Revision     : 04
Sensor Type      : Power Supply
Sensor Number    : 02
Event Type       : Sensor-specific Discrete
Event Direction  : Deassertion Event
Event Data       : 00ffff
Description      : Presence detected

SEL Record ID    : 001b
Record Type      : 02
Timestamp        : 08/06/2013 15:05:21
Generator ID     : 0044
EvM Revision     : 04
Sensor Type      : Power Supply
Sensor Number    : 03
Event Type       : Sensor-specific Discrete
Event Direction  : Assertion Event
Event Data       : 01ffff
Description      : Failure detected

SEL Record ID    : 001c
Record Type      : 02
Timestamp        : 08/06/2013 15:05:22
Generator ID     : 0044
EvM Revision     : 04
Sensor Type      : Power Supply
Sensor Number    : 04
Event Type       : Sensor-specific Discrete
Event Direction  : Assertion Event
Event Data       : 01ffff
Description      : Failure detected

SEL Record ID    : 001d
Record Type      : 02
Timestamp        : 08/06/2013 15:05:22
Generator ID     : 0044
EvM Revision     : 04
Sensor Type      : Power Supply
Sensor Number    : 05
Event Type       : Sensor-specific Discrete
Event Direction  : Assertion Event
Event Data       : 01ffff
Description      : Failure detected

SEL Record ID    : 001e
Record Type      : 02
Timestamp        : 08/06/2013 15:05:26
Generator ID     : 0044
EvM Revision     : 04
Sensor Type      : Power Supply
Sensor Number    : 02
Event Type       : Sensor-specific Discrete
Event Direction  : Assertion Event
Event Data       : 01ffff
Description      : Failure detected

SEL Record ID    : 001f
Record Type      : 02
Timestamp        : 08/09/2013 14:34:48
Generator ID     : 0044
EvM Revision     : 04
Sensor Type      : Fan
Sensor Number    : 07
Event Type       : Generic Discrete
Event Direction  : Assertion Event
Event Data       : 04ffff
Description      : Transition to Off Line

```

SEL Record ID : 0020
Record Type : 02
Timestamp : 08/09/2013 14:34:49
Generator ID : 0044
EvM Revision : 04
Sensor Type : Fan
Sensor Number : 07
Event Type : Generic Discrete
Event Direction : Deassertion Event
Event Data : 00ffff
Description : Transition to Running

SEL Record ID : 0021
Record Type : 02
Timestamp : 08/09/2013 14:34:53
Generator ID : 0044
EvM Revision : 04
Sensor Type : Fan
Sensor Number : 07
Event Type : Generic Discrete
Event Direction : Assertion Event
Event Data : 00ffff
Description : Transition to Running

SEL Record ID : 0022
Record Type : 02
Timestamp : 08/09/2013 14:35:11
Generator ID : 0044
EvM Revision : 04
Sensor Type : Fan
Sensor Number : 09
Event Type : Generic Discrete
Event Direction : Assertion Event
Event Data : 04ffff
Description : Transition to Off Line

SEL Record ID : 0023
Record Type : 02
Timestamp : 08/09/2013 14:35:11
Generator ID : 0044
EvM Revision : 04
Sensor Type : Fan
Sensor Number : 09
Event Type : Generic Discrete
Event Direction : Deassertion Event
Event Data : 00ffff
Description : Transition to Running

SEL Record ID : 0024
Record Type : 02
Timestamp : 08/09/2013 14:35:16
Generator ID : 0044
EvM Revision : 04
Sensor Type : Fan
Sensor Number : 09
Event Type : Generic Discrete
Event Direction : Assertion Event
Event Data : 00ffff
Description : Transition to Running

SEL Record ID : 0025
Record Type : 02
Timestamp : 08/09/2013 14:35:21
Generator ID : 0044
EvM Revision : 04
Sensor Type : Fan
Sensor Number : 08
Event Type : Generic Discrete
Event Direction : Assertion Event
Event Data : 04ffff
Description : Transition to Off Line

SEL Record ID : 0026
Record Type : 02
Timestamp : 08/09/2013 14:35:21
Generator ID : 0044
EvM Revision : 04
Sensor Type : Fan
Sensor Number : 08
Event Type : Generic Discrete
Event Direction : Deassertion Event

```

Event Data          : 00ffff
Description         : Transition to Running

SEL Record ID      : 0027
Record Type        : 02
Timestamp          : 08/09/2013 14:36:09
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Fan
Sensor Number      : 09
Event Type         : Generic Discrete
Event Direction    : Assertion Event
Event Data         : 04ffff
Description        : Transition to Off Line

SEL Record ID      : 0028
Record Type        : 02
Timestamp          : 08/09/2013 14:36:09
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Fan
Sensor Number      : 09
Event Type         : Generic Discrete
Event Direction    : Deassertion Event
Event Data         : 00ffff
Description        : Transition to Running

SEL Record ID      : 0029
Record Type        : 02
Timestamp          : 08/09/2013 14:36:14
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Fan
Sensor Number      : 09
Event Type         : Generic Discrete
Event Direction    : Assertion Event
Event Data         : 00ffff
Description        : Transition to Running

SEL Record ID      : 002a
Record Type        : 02
Timestamp          : 08/09/2013 14:36:31
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Fan
Sensor Number      : 08
Event Type         : Generic Discrete
Event Direction    : Assertion Event
Event Data         : 00ffff
Description        : Transition to Running

SEL Record ID      : 002b
Record Type        : 02
Timestamp          : 08/09/2013 14:36:45
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Fan
Sensor Number      : 0a
Event Type         : Generic Discrete
Event Direction    : Assertion Event
Event Data         : 04ffff
Description        : Transition to Off Line

SEL Record ID      : 002c
Record Type        : 02
Timestamp          : 08/09/2013 14:36:45
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Fan
Sensor Number      : 0a
Event Type         : Generic Discrete
Event Direction    : Deassertion Event
Event Data         : 00ffff
Description        : Transition to Running

SEL Record ID      : 002d
Record Type        : 02
Timestamp          : 08/09/2013 14:36:50
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Fan

```

```

Sensor Number      : 0a
Event Type         : Generic Discrete
Event Direction    : Assertion Event
Event Data         : 00ffff
Description        : Transition to Running

SEL Record ID     : 002e
Record Type        : 02
Timestamp          : 08/09/2013 14:36:52
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Fan
Sensor Number      : 0b
Event Type         : Generic Discrete
Event Direction    : Assertion Event
Event Data         : 04ffff
Description        : Transition to Off Line

SEL Record ID     : 002f
Record Type        : 02
Timestamp          : 08/09/2013 14:36:52
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Fan
Sensor Number      : 0b
Event Type         : Generic Discrete
Event Direction    : Deassertion Event
Event Data         : 00ffff
Description        : Transition to Running

SEL Record ID     : 0030
Record Type        : 02
Timestamp          : 08/09/2013 14:36:57
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Fan
Sensor Number      : 0b
Event Type         : Generic Discrete
Event Direction    : Assertion Event
Event Data         : 00ffff
Description        : Transition to Running

SEL Record ID     : 0031
Record Type        : 02
Timestamp          : 08/09/2013 14:43:23
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Fan
Sensor Number      : 08
Event Type         : Generic Discrete
Event Direction    : Assertion Event
Event Data         : 04ffff
Description        : Transition to Off Line

SEL Record ID     : 0032
Record Type        : 02
Timestamp          : 08/09/2013 14:43:23
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Fan
Sensor Number      : 08
Event Type         : Generic Discrete
Event Direction    : Deassertion Event
Event Data         : 00ffff
Description        : Transition to Running

SEL Record ID     : 0033
Record Type        : 02
Timestamp          : 08/09/2013 14:43:54
Generator ID       : 0044
EvM Revision       : 04
Sensor Type        : Fan
Sensor Number      : 08
Event Type         : Generic Discrete
Event Direction    : Assertion Event
Event Data         : 00ffff
Description        : Transition to Running

SEL Record ID     : 0034
Record Type        : 02
Timestamp          : 08/09/2013 14:49:46

```

```

Generator ID      : 0044
EvM Revision     : 04
Sensor Type      : Fan
Sensor Number    : 0a
Event Type       : Generic Discrete
Event Direction  : Assertion Event
Event Data       : 04ffff
Description      : Transition to Off Line

SEL Record ID    : 0035
Record Type      : 02
Timestamp        : 08/09/2013 14:49:46
Generator ID     : 0044
EvM Revision     : 04
Sensor Type      : Fan
Sensor Number    : 0a
Event Type       : Generic Discrete
Event Direction  : Deassertion Event
Event Data       : 00ffff
Description      : Transition to Running

SEL Record ID    : 0036
Record Type      : 02
Timestamp        : 08/09/2013 14:49:55
Generator ID     : 0044
EvM Revision     : 04
Sensor Type      : Fan
Sensor Number    : 0a
Event Type       : Generic Discrete
Event Direction  : Assertion Event
Event Data       : 00ffff
Description      : Transition to Running

SEL Record ID    : 0037
Record Type      : 02
Timestamp        : 08/09/2013 14:50:12
Generator ID     : 0044
EvM Revision     : 04
Sensor Type      : Fan
Sensor Number    : 08
Event Type       : Generic Discrete
Event Direction  : Assertion Event
Event Data       : 04ffff
Description      : Transition to Off Line

SEL Record ID    : 0038
Record Type      : 02
Timestamp        : 08/09/2013 14:50:12
Generator ID     : 0044
EvM Revision     : 04
Sensor Type      : Fan
Sensor Number    : 08
Event Type       : Generic Discrete
Event Direction  : Deassertion Event
Event Data       : 00ffff
Description      : Transition to Running

SEL Record ID    : 0039
Record Type      : 02
Timestamp        : 08/09/2013 14:50:31
Generator ID     : 0044
EvM Revision     : 04
Sensor Type      : Fan
Sensor Number    : 08
Event Type       : Generic Discrete
Event Direction  : Assertion Event
Event Data       : 00ffff
Description      : Transition to Running

SEL Record ID    : 003a
Record Type      : 02
Timestamp        : 08/09/2013 15:00:54
Generator ID     : 0044
EvM Revision     : 04
Sensor Type      : Fan
Sensor Number    : 0b
Event Type       : Generic Discrete
Event Direction  : Assertion Event
Event Data       : 04ffff
Description      : Transition to Off Line

```


SEL Record ID : 003b
Record Type : 02
Timestamp : 08/09/2013 15:00:54
Generator ID : 0044
EvM Revision : 04
Sensor Type : Fan
Sensor Number : 0b
Event Type : Generic Discrete
Event Direction : Deassertion Event
Event Data : 00ffff
Description : Transition to Running

SEL Record ID : 003c
Record Type : 02
Timestamp : 08/09/2013 15:01:03
Generator ID : 0044
EvM Revision : 04
Sensor Type : Fan
Sensor Number : 0b
Event Type : Generic Discrete
Event Direction : Assertion Event
Event Data : 00ffff
Description : Transition to Running

SEL Record ID : 003d
Record Type : 02
Timestamp : 08/09/2013 15:01:30
Generator ID : 0044
EvM Revision : 04
Sensor Type : Fan
Sensor Number : 08
Event Type : Generic Discrete
Event Direction : Assertion Event
Event Data : 04ffff
Description : Transition to Off Line

SEL Record ID : 003e
Record Type : 02
Timestamp : 08/09/2013 15:01:30
Generator ID : 0044
EvM Revision : 04
Sensor Type : Fan
Sensor Number : 08
Event Type : Generic Discrete
Event Direction : Deassertion Event
Event Data : 00ffff
Description : Transition to Running

SEL Record ID : 003f
Record Type : 02
Timestamp : 08/09/2013 15:02:05
Generator ID : 0044
EvM Revision : 04
Sensor Type : Fan
Sensor Number : 08
Event Type : Generic Discrete
Event Direction : Assertion Event
Event Data : 00ffff
Description : Transition to Running

Glossary

ACPI	Advanced Configuration and Power Interface Specification
BCD	Binary-coded Decimal
BMC	Baseboard Management Controller
BT	Block Transfer
ChMC	Chassis Management Controller
CMOS	The PC-AT compatible region of battery-backed 128 bytes of memory, which normally resides on the baseboard
CTS	Clear to send
DCD	Data Carrier Detect
DCMI	Data Center Manageability Interface
DSR	Data Set Ready
DTR	Data Transfer Request
EvMRev	Event message revision
FPGA	Field-Programmable Gate Array
FRB	Fault-resilient booting
FRU	Field replaceable unit
GUID	Globally Unique ID
HA	High availability
I²C	Inter-Integrated Circuit
IANA	Internet Assigned Numbers Authority
ICMB	Intelligent Chassis Management Bus
IERR	Internal error
IPMB	Intelligent Platform Management Bus
IPMI	Intelligent Platform Management Interface
KCS	Keyboard Controller Style
LPC	Low Pin Count
LS	Least significant byte
MC	Management Controller
MS	Most significant byte
mux	multiplexing
NAK	Negative-acknowledge character
NetFn	Network Function
NMI	Non-maskable Interrupt
PCI	Peripheral Component Interconnect
PEF	Platform Event Filtering
PICMG	PCI Industrial Computer Manufacturers Group
POH	Power-On Hours
PSMC	Power Supply Management Controller
RAKP	Remote Authenticated Key-Exchange Protocol
RMCP	Remote Management Control Protocol
RQ	Received request
RS	Received response

rsSA	Random Single Switch Algorithm
RTS	Request to send
SCI	Software Configuration Identification
SDR	Sensor Data Record
SEL	System Event Log
SMB	System Management Bus
SMI	System Management Interrupt
SMIC	System Management Interface Chip
SMM	System Management Mode.
SMS	System Management Software
SOL	Serial Over LAN
SSI	Server System Infrastructure
SSIF	SMBus System Interface
SWID	Software ID
TAP	Telocator Access Protocol
UUID	Universally Unique IDentifier
VSO	VITA Standards Organization

Index

A

ACK messages, 155
active sessions, 57
authcode, 59
authentication, 144
authorized resellers, 157

B

BMC, 141
 interfaces, 141
BT, 19

C

channel access, 66
channel accessibility, 65
channel number, 144
chassis status, 22
command line tool, 18
command set, 141
command value, 141
Commands
 Get Sensor Reading, 7
commands
 Activate Session, 144
 activate session, 56
 add sel entry, 114
 CMD, 141
 get channel access limits, 65
 Get Channel Info, 145
 Get Device ID, 17
 Get SDR, 11
 Get SDR Repository Info, 11
 Get Session, 144
 Master Write-Read, 12
 Read/Write FRU, 12
 SEEPROM, 12
 set channel access, 56
 set user access, 56
 verbose, 21
completion code, 142
completion condition, 142
configure channels, 60
contacting HP, 157

D

data block, 59
device discovery, 17
dynamic controllers, 17

E

event receiver, 114
events, 21

F

FRU, 11

data, 11
 inventory device, 12
function codes, 141
function command, 140

H

handshake
 per block, 140
 per byte, 140
help
 obtaining, 157
HP
 technical support, 157

I

ICMB, 141
inventory
 remote, 22
IPMI
 capabilities, 7
 definition, 7
IPMI LAN session, 156
IPMItool, 18
 definition, 18
 discovery of features, 20
 encryption, 19
 LANplus interface, 19
 remote chassis power control, 18
IPMV, 141
IPv4, 19

K

KCS, 140, 141

L

LAN, 141
LANPlus interface
 RMCP+, 19

Linux

 kernel driver, 19
 OpenIPMI, 23

M

message interface, 141
message payload, 140
microcontroller, 140

N

NetFn, 140
network function code, 140

O

OpenIPMI, 19
OpenSSL, 19

P

- parsing hierarchy, 141
- payload commands, 70
- POH counter, 12
- print field replaceable unit, 18
- privilege level, 65

R

- RCMP+, 70
- remote systems, 22
- response messages, 142
- RMCP, 154, 155
- RMCP+ protocol, 156

S

- SDR
 - format, 10
 - platform management, 10
 - purpose, 10
 - record body, 10
 - record header, 10
 - record key, 10
 - repository, 10
 - repository device, 12
- SEL, 114
 - SEL commands, 114
 - SEL device, 114
- SEL commands, 9
- Sensor data model, 7
- sensor data repository, 18
- sensor owner
 - sensor owner id, 7
- sensor type codes, 8
- serial/modem, 141
- session, 144
- session-less, 144
- SMCI, 141
- SOL, 156
- Solaris
 - BMC, 23
- SSIF, 140
- Support and other resources, 157
- system interface, 19

T

- technical support
 - HP, 157
- timers
 - POH counter, 12
 - Watchdog, 12
- timestamp
 - format, 12
 - values, 12

U

- user id, 69
- user password, 69

V

- virtual chassis manager, 17